Journal of Research and Practice in Information Technology

Volume 32
number 2 2000

JOURNAL OF RESEARCH AND PRACTICE IN INFORMATION TECHNOLOGY
Formerly the Australian Computer Journal

The *Journal of Research and Practice in Information Technology* is a quarterly publication emphasising activities that have successfully connected fundamental and applied research in Information Technology with practical application. The journal publishes papers relating to both emerging research and to professional practice and thus contains articles that are of interest both to practicing information technology professionals and to university and industry researchers.

Editor (and address for correspondence):
Professor John F. Roddick
Editor-in-Chief
Journal of Research and Practice in Information Technology
Flinders University of South Australia
GPO Box 2100
Adelaide 5001, South Australia

Associate Editors
Associate Professor Graham Low
Associate Editor – Information Systems
The University of New South Wales

Professor Maria Orlowska
Associate Editor – Database Systems
The University of Queensland

Professor Bernhard Thalheim
Associate Editor – Conceptual Modelling
Brandenburg Technical University at Cottbus, Germany.

Dr Hugh Williams
Associate Editor – Computer Science
RMIT University

Denis Warne
Associate Editor – Software Engineering
BAE SYSTEMS

David Williams
Books Editor
The University of Newcastle

Professor Ian Witten
Associate Editor – Digital Libraries
University of Waikato, New Zealand.

Tom Worthington
Director, Publications Board
Australian National University

Office Bearers
J. Ridge, President
D. Street, Vice President
P. Kalkman, Vice President
P. Ralston, Immediate Past President
R.G. Heinrich, National Treasurer
D.W. Furini, Chief Executive
PO Box Q534, QVB Post office, Sydney, NSW 1230, Australia
Telephone: (02) 9299 3666
Facsimile: (02) 9299 3997
email: info@acs.org.au
URL: www.acs.org.au

Copyright© 2000, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the JRPIT's copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by the Australian Computer Society Inc.

Print Post Approved PP255003/01660

Welcome to the second issue of the new-look *Journal of Research and Practice in Information Technology* (or JRPIT as everyone seems to have abbreviated it to). We are extremely pleased by the way the new format and style has been received and the internationalisation of the Journal is resulting in considerable interest worldwide. In particular, the journal is now being abstracted by an increasing number of agencies and the papers being submitted are now clearly international in flavour.

This issue contains four high quality papers covering a wide range of issues and interests including the partitioning of number sequences, the partitioning of objects in distributed applications, the slicing of Petri net models and a discussion on the application of the ACS Code of Ethics.

In addition, we have recently announced two special issues of the Journal for 2001:

- *Distributed Systems*, to be edited by Michael Oudshoorn from Adelaide University,
- *Web-Based Information Systems*, to be edited by Michael Schrefl from Johannes Kepler University, Austria, and myself.

Full details, including the call for papers, can be found at the Journal's website, *http://www.jrpit.flinders.edu.au.*

Also now available from the website, thanks to the hard work of Godfrey Lance, a former editor of the Journal from 1968 to 1971, is a complete listing, with abstracts, of every paper published since the *Australian Computer Journal* commenced publication in 1967. The website also provides online, the text of the first page of all papers published since the change in format.

It is also a pleasure for me to introduce the editorial team that will be helping me with the Journal over the next few years. They act as the central point of contact with the referees and with the volume of papers being submitted now increasing, their workload is not inconsiderable. Some of them will be familiar to past readers of the ACJ and some are new for 2000.

Associate Professor Graham Low
Associate Editor – Information Systems
The University of New South Wales

Professor Maria Orlowska
Associate Editor – Database Systems
The University of Queensland

Professor Bernhard Thalheim
Associate Editor – Conceptual Modelling
Brandenburg Technical University at Cottbus, Germany.

Dr Hugh Williams
Associate Editor – Computer Science
RMIT University

Denis Warne
Associate Editor – Software Engineering
BAE SYSTEMS

David Williams
Books Editor
The University of Newcastle

Professor Ian Witten
Associate Editor – Digital Libraries
University of Waikato, New Zealand.

Many other exciting developments to the Journal are underway (some can be found at the website) and more changes can be expected in the next few months. As you would expect, we are endeavouring to continually improve the publication and we would welcome feedback and any comments you might have.

John Roddick
Flinders University

Partitioning and Allocation of Objects in Distributed Application Development

G.C. Low

School of Information Systems
The University of New South Wales
Sydney, NSW 2052 Australia
email: G.Low@unsw.edu.au

G. Rasmussen

Currently at Poynton and Partners, Perth, Australia

In recent years there has been significant interest in the use of object-oriented techniques for the production of distributed systems. However, only a limited amount of work has been reported on the incorporation of distributed system design issues into object-oriented development methodologies. The identification of processes within a software system (partitioning) and the allocation of these processes to processors in the system is a fundamental problem in the design of distributed systems.

Techniques for evaluating the three objectives of task partitioning: minimizing inter-module communications, exploiting concurrency, and limiting the size of processes are presented. The potential concurrency in the object model is graded into three categories. This grading of the potential concurrency between objects/classes is new and is anticipated may assist in the partitioning decision making process. Communication and execution costs required for software allocation are determined from the event diagrams and event traces. The aim of the paper is to use the object-oriented model to produce information that fits into existing non object-oriented decision techniques such as traditional graph theoretic or graph heuristic allocation techniques.

CR Categories: Object-oriented design methods and software engineering practices; distributed systems.

1. INTRODUCTION

In recent years there has been significant interest in the use of object-oriented techniques for the production of distributed systems. However, the majority of work has been at the implementation level. Several distributed object-oriented programming languages (Black *et al,* 1987) have been developed, as well as distributed object-oriented operating systems (Bennett, 1987) and systems to provide distribution services to object-oriented applications (Atkinson, 1991). Only a limited amount of work has been reported on the incorporation of distributed system design issues into object-oriented development methodologies. Low *et al,* (1996) discussed the modifications required, in general terms, for a generic OO development methodology. Rasmussen *et al,* (1996a) applied this work to the MOSES methodology.

Copyright© 2000, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the JRPIT copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

Manuscript received: May 2000
Guest Editor: Brian Henderson-Sellers

A system that is to be distributed around a network must be broken down into components that are allocated to physical nodes. The process of breaking the system down into those components is called *partitioning* where the object model is partitioned into cohesive and independent components based on requirements for concurrency, communications, and limiting the size of processes. These are then allocated to one or more processors with the actual *allocation* being dependent on issues such as interprocess communication cost, execution cost, load balancing, reliability and scalability whilst not violating constraints such as memory size limits, processing capacity limits, required allocations and disallowed allocations (Shatz, 1993).

The allocation of software components to the nodes in a network is a problem of exponential complexity. Under the assumption that processes (components) are allocated to only one physical node, an m process and n processor problem has n^m possible allocation solutions. In reality, the replication of components and dynamic allocation must also be considered, making the problem even more complicated (Shatz, 1993).

Chu *et al,* (1980) identified three specific types of approaches to the allocation decision problem. These are mathematical programming (Gylys and Edwards, 1976; Lee and Muntz, 1977), graph theoretic (Jenny, 1977; Stone and Bokhari, 1978) and heuristic (Gylys and Edwards, 1976; Efe, 1982; Ezzat *et al,* 1986). The key to each of these types of approach is an understanding of the amount of communications between components, and the amount of processing required for each component.

This paper describes the use the object-oriented model to produce information which fits into non object-oriented decision techniques. In particular it considers:

- partitioning. Section 2 briefly reviews key aspects of proposed notation enhancements to an existing object-oriented methodology (MOSES – Rasmussen *et al,* (1996b)) to support distributed application design. These enhancements are reflected in the object models which are used to illustrate the partitioning and allocation process presented in this paper. Section 3 then provides guidelines for the derivation of information on the communications between objects (modules), the potential for concurrency between objects (modules), and size of software components; and
- software allocation. Sections 4 provides guidelines for the derivation of information for two of the possible allocation cost metrics – interprocess communications, and interprocess communications plus execution cost. Ongoing research is considering other allocation cost metrics.

2. DISTRIBUTED SYSTEMS NOTATION EXTENSIONS

Rasmussen *et al,* (1996b) extended the MOSES notation to support the design of distributed object-oriented systems. Task partitioning and allocation, interprocess communications and concurrency control semantics were added to the original notation. These extensions permitted sophisticated allocation of objects (either as individuals or as classes) to nodes (virtual and physical), incorporation of synchronous, asynchronous and mutually exclusive message invocations and a multi-threaded environment by use of extended Event Models and Objectcharts. A brief review of the relevant aspects is presented since many of the concepts are reflected in the object models which are used to illustrate the partitioning and allocation process presented in this paper.

2.1 Asynchronous and synchronous communications

From the client object perspective, communication may be synchronous or asynchronous. When it is synchronous, the client is blocked and may only continue when the server completes processing the service request. A synchronous service request is shown with a solid square at the base of the

communication line in the Event Model (Figure 1a). Asynchronous communication allows the client to continue processing after it sends a message requesting a service. These messages are shown with an open square at the base of the line representing the message (Figure 1b).

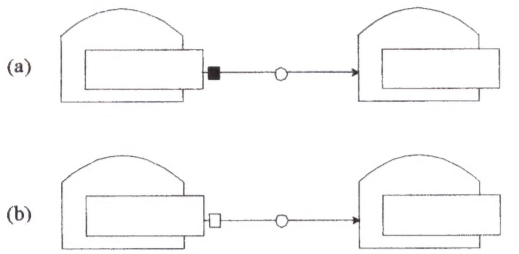


Figure 1: – (a) Synchronous communications
(b) Asynchronous communications

2.2 Representation of partitioning and allocation decisions

Virtual nodes are included in the O/C[1] model to represent partitioning decisions and may be linked to physical nodes to show allocation decisions. Virtual nodes, which represent a software process, have an octagon as their symbol as used by Mühlhaüser *et al,* (1993). These processes are the output of the partitioning process. Unfilled octagons represent a lightweight process (Figure 2a) which is a process that must be started by another process rather than executed on its own. Filled octagons represent a heavyweight process or processes (Figure 2b). These may be executed on their own.

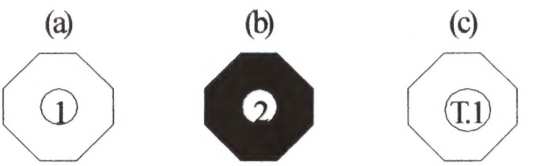


Figure 2: Symbols for a virtual node: (a) Lightweight
(b) Heavyweight (c) Allocated to physical node with code T

The virtual nodes are all numbered to allow unique identification during design. If they have been allocated to a physical node as a result of the allocation process, then a letter representing that physical node is placed before the number in the virtual node. For example, in Figure 2(c), the virtual node has been allocated to the processor with code "T". The code is separated from the number using a dot and may represent an individual physical node or a type of physical node. The virtual nodes in Figure 2(a) and Figure 2(b) have not yet been allocated to physical nodes, as no letter is shown.

An object/class, several objects/classes or a subsystem full of objects/classes may be allocated to a virtual node in the O/C model. This may be done in two ways. The first is by drawing a dashed line from the virtual node to each of the objects/classes and subsystems allocated to it. Alternatively, a dotted line may be drawn around a group of objects/classes that are to be allocated to a virtual node. The surrounding line should then be linked to the virtual node. Figure 3 shows

[1] The O/C Model describes the inheritance, association and aggregation relationships that exist between classes.

both methods of showing the allocation decision. O/Cs A and B are allocated to the virtual node with code 1. Note that B is an active object/class, which allows the use of a heavyweight virtual node. O/Cs C and D are allocated to virtual node with code 2. Finally, subsystem E is allocated to the virtual node number 3.

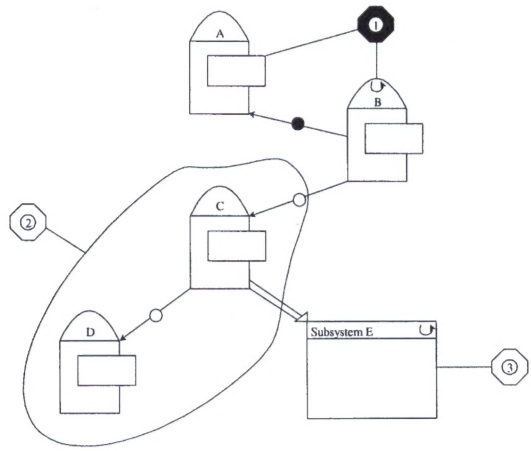


Figure 3: Allocation of objects/classes and subsystems to virtual nodes

A virtual node, combined with the objects/classes that have been allocated to it, is equivalent to a distributed processing component (DPC). These distributed processing components are discussed by authors such as Shatz and Yau (1986) in the distributed systems partitioning and task allocation literature. DPCs are essentially processes.

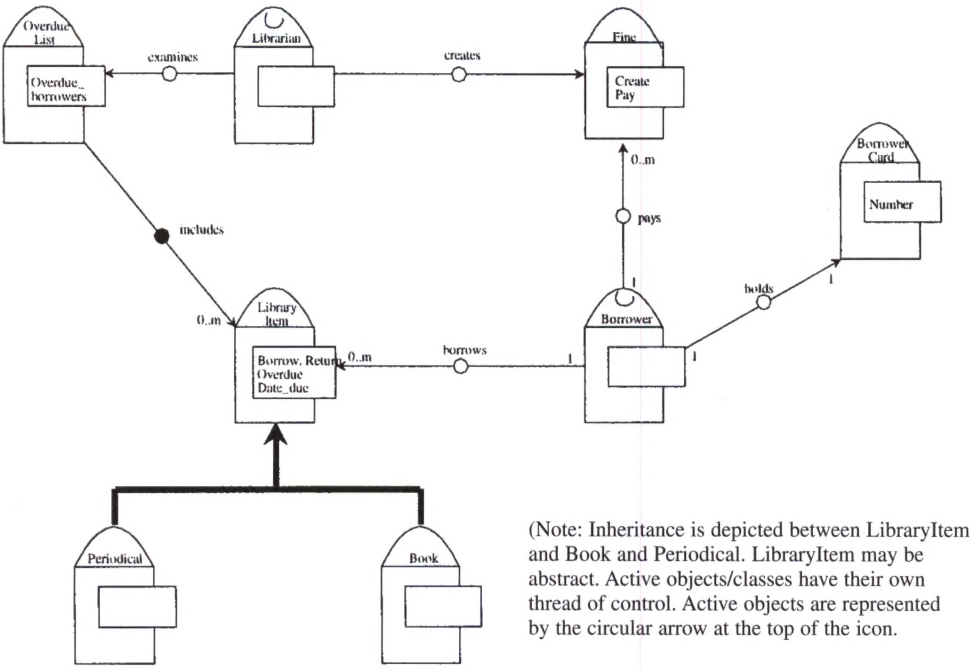
3. PARTITIONING

Section 3.1 presents an example system used to illustrate the partitioning process. Sections 3.2 to 3.4 provide guidelines for the derivation of information on the communications between objects (modules), the potential for concurrency between objects (modules), and size of software components. The aim of this section on partitioning is to use the object-oriented model to produce information which fits into non object-oriented decision techniques.

3.1 Example system

Consider the system shown in Figure 4. It supports some of the functionality expected of a library system - borrowing, returning and charging fines for overdue books. The O/C model shows that there are eight objects/classes that need to be partitioned into one or more software components. The event models in Figures 5, 6, 7 and 8 depict the events that the system must handle. Tables 1, 2, 3 and 4 show the event traces for these event model diagrams.

A borrower has a card which holds the borrower number. He/she may borrow and return library items which may be books or periodicals. The library items have a due date when they are on loan and are overdue when the current date goes beyond that date. Once a day the librarian will create an overdue list, which will query all books on loan and build a list of the borrowers who have overdue books and the date on which those books were due. The librarian then creates fines for the overdue books, which the borrowers must pay before borrowing more books.



(Note: Inheritance is depicted between LibraryItem and Book and Periodical. LibraryItem may be abstract. Active objects/classes have their own thread of control. Active objects are represented by the circular arrow at the top of the icon.

Figure 4: O/C model for a library system

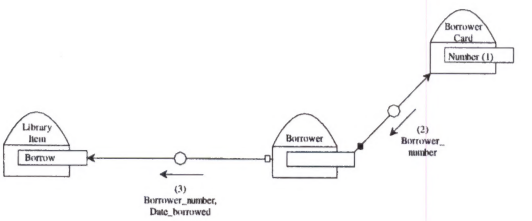


Figure 5: Event model for borrowing an item

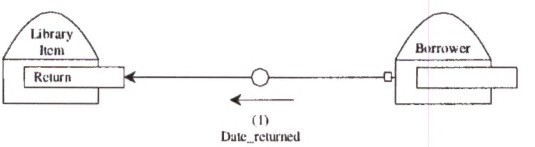


Figure 6: Event model for returning an item

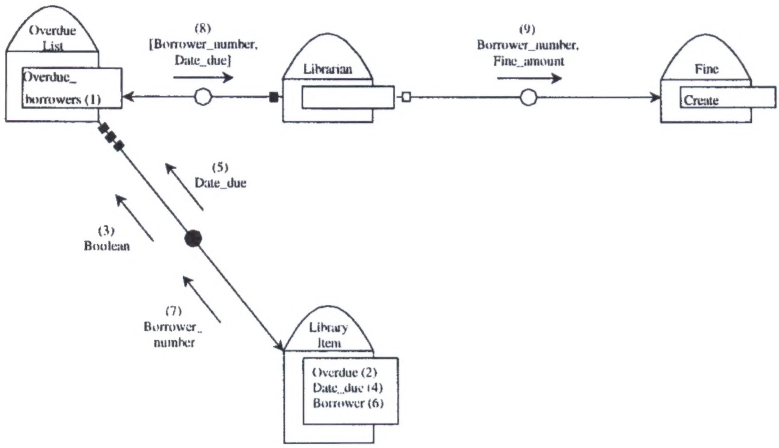


Figure 7: Event model for creating overdue item fines

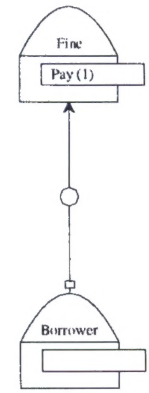


Figure 8: Event model for paying a fine

When a borrower wishes to borrow a book (Figure 5), the borrower number is retrieved from his/her card, through the Number service. The borrower then requests the Borrow service of the library item desired, passing the borrower number and date of borrowing as arguments. The library item marks itself as on loan.

When a book is returned (Figure 6), the borrower requests the Return service in the library item. The date of return is passed as an argument to that service.

When the librarian wishes to create fines for overdue books (Figure 7), an overdue list object is created. The overdue list object queries all library items on loan to see if they are overdue or not using the Overdue service. For those that return TRUE, the overdue list requests the date due and borrower number. This information is then aggregated and returned to the librarian object, which creates a fine for each borrower, calculating the amount owed using the date the book was due.

When a borrower wishes to pay the fine (Figure 8), the borrower object sends a message to the fine object, requesting the Pay service. This deletes the fine object.

3.2 Communications

The level of communication between each object/class can be determined from the event model diagrams and their accompanying trace tables. Possible measures of communication between

Borrow an Item		*200 per day*			
Msg	**From**	**To**	**Service Requested**	**Argument**	**Repeat**
1	Borrower	Borrower Card	Number		1
2	Borrower Card	Borrower	RETURN	Borrower_number	1
3	Borrower	Library Item	Borrow	Borrower_number, Date_borrowed	1

(Note: RETURN represents the service in the client class awaiting the response from the server class. In the above table the server class is Borrower Card)

Table 1: Trace for event model in Figure 5

Borrow an Item		*200 per day*			
Msg	**From**	**To**	**Service Requested**	**Argument**	**Repeat**
1	Borrower	Borrower Card	Number		1
2	Borrower Card	Borrower	RETURN	Borrower_number	1
3	Borrower	Library Item	Borrow	Borrower_number, Date_borrowed	1

Table 2: Trace for event model in Figure 6

Create overdue fines		*1 per day*			
Msg	**From**	**To**	**Service Requested**	**Argument**	**Repeat**
1	Librarian	Overdue List	Overdue borrowers		1
2	Overdue list	Library Item	Overdue		300
3	Library Item	Overdue List	RETURN	BOOLEAN	300
4	Overdue List	Library Item	Date due		10
5	Library Item	Overdue List	RETURN	Date_due	10
6	Overdue List	Library Item	Borrower		10
7	Library Item	Overdue List	RETURN	Borrower_number	10
8	Overdue List	Librarian	RETURN	LIST (Borrower_number, Date_due)	1
9	Librarian	Fine	Create	Borrower number, Fine_amount	10

(Note: LIST is the list of all borrowers who have overdue books and the date on which those books were due)

Table 3: Trace for event model in Figure 7

objects/classes include:

- counting the number of messages sent;
- counting the number of messages sent and weighting these messages based on the number of arguments sent with the message; and
- counting the number of messages sent and weighting these messages based on the size of arguments sent.

It is likely that the simple method is enough at the time of partitioning. Indeed, the third method requires knowledge of the implementation details for each object/class, which may not be available at the time of partitioning.

Borrow an Item		*200 per day*			
Msg	**From**	**To**	**Service Requested**	**Argument**	**Repeat**
1	Borrower	Borrower Card	Number		1
2	Borrower Card	Borrower	RETURN	Borrower_number	1
3	Borrower	Library Item	Borrow	Borrower_number, Date_borrowed	1

Table 4: Trace for event model in Figure 8

Tables 1 to 4 show the event traces for the library system. For each event, the estimated number of times it will occur in one day is shown. For each object/chart interaction, the number of times the message is expected to be sent from one object to another (within one event) is shown in the "Repeat" column. The values in this column are summed for tuples that show messages between the same objects/classes to the total number of messages between the two objects/classes.

	Borrower	**Borrower Card**	**Fine**	**Librarian**	**Library Item**	**Overdue List**
Borrower	0	400	10	0	400	0
Borrower Card	400	0	0	0	0	0
Fine	10	0	0	10	0	0
Librarian	0	0	10	0	0	2
Library Item	400	0	0	0	0	640
Overdue List	0	0	0	2	640	0

Table 5: Calculations of the communication levels between objects/classes

Table 5 shows the number of messages each day between the various objects/classes. The figures from Table 5 are shown on the O/C model diagram for the library system (Figure 9). Note that the messages being sent to and from Book and Periodical are not distinguished. This shows an assumption that they will be allocated to the same virtual node. This assumption is not necessary but was done here to simplify the example.

3.3 Identification of Concurrency

Potential for concurrency can be seen in the O/C and event model diagrams. The O/C model diagrams may be used for dependency based techniques to identify objects/classes that may operate concurrently. These techniques may then be refined using the event models, by looking for asynchronous interaction between objects/classes, and examining the dependencies at a lower level of granularity.

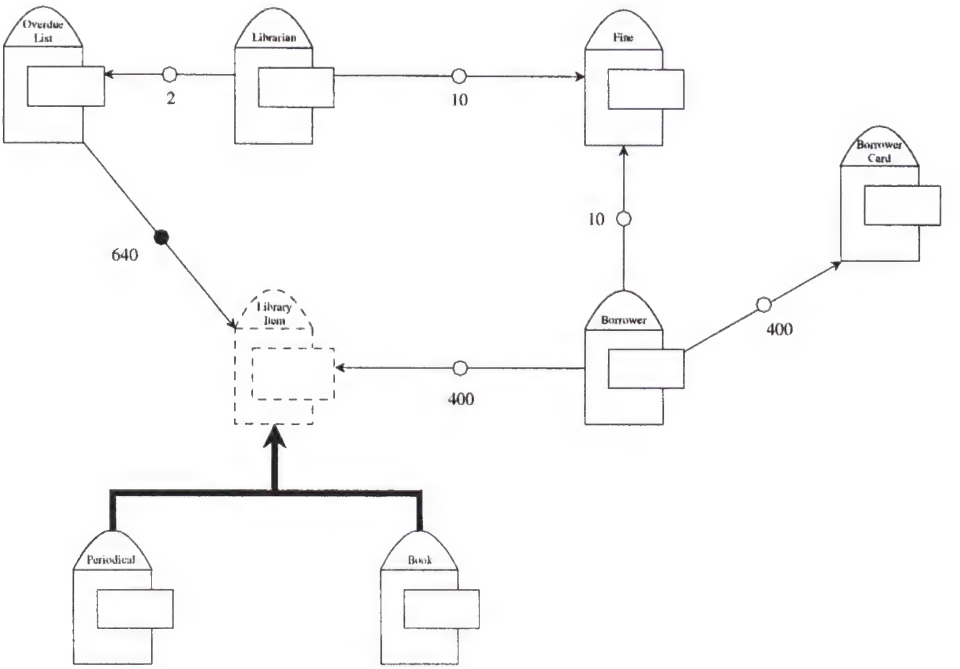


Figure 9: Showing the number of messages between objects/classes for the partitioning decision

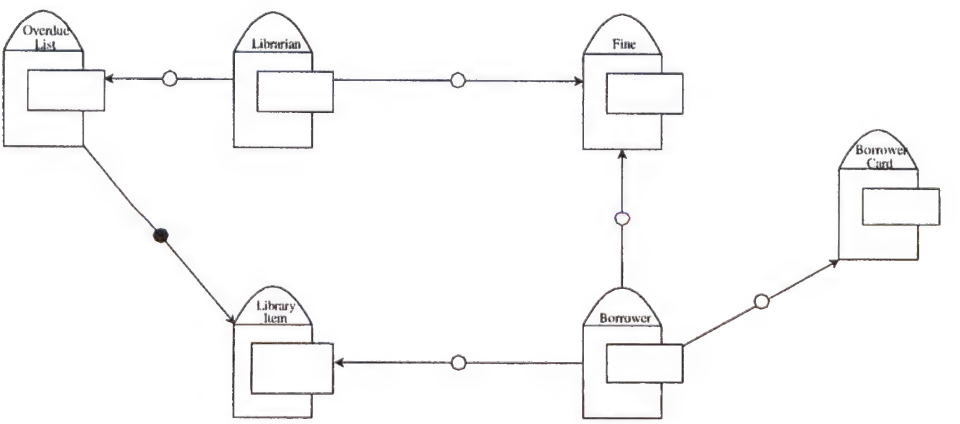


Figure 10: O/C Model to be treated as a directed graph

	Borrower	Borrower Card	Fine	Librarian	Library Item	Overdue List
Borrower	X	X	X	-	X	-
Borrower Card	X	X	-	-	-	-
Fine	X	-	X	X	-	-
Librarian	-	-	X	X	-	X
Library Item	X	-	-	-	X	X
Overdue List	-	-	-	X	X	X

Table 6: Adjacency matrix

Independence of modules is seen by the absence of a control path between them in the structure decomposition case. This rule may be applied to the object model, using the O/C model notation. Figure 10 is a replication of the O/C model diagram from Figure 4, with some detail removed. If the objects/classes are treated as nodes on a directed graph, then the transitive closure of that graph may be used to see the dependencies between objects/classes. An adjacency matrix (Table 6) may be made for the O/C model, where a "X" represents that there is a path from one object/class (represented by the rows in the matrix) to a neighbour object/class (represented by the columns in the matrix). As objects/classes cannot operate concurrently with themselves, these entries in the matrix are also be set to "X". Warshall's algorithm (Floyd, 1962) may then be used for the matrix to find the transitive closure (Table 7).

```
Warshall's Algorithm [O(n2)]
    — n is the number of nodes in the directed graph
    — A is a matrix representing the graph
    — if A[j,i] = X then there is a path from j to i
    — start as an adjacency matrix
    FOR i := 1 TO n DO
            FOR j := 1 TO n DO
                    IF A[ j,i ] = 'X' THEN
                            FOR k := 1 TO n DO
                                    IF A[ i,k ] = 'X' THEN
                                            A[ j,k ] := 'X'
                                    END;
                            END;
                    END;
            END;
    END;
```

In the transitive closure matrix (Table 7), objects/classes with "X"s between them are considered to be dependent on each other. Figure 11 shows how the transitive closure looks in an adaptation of the O/C model notation. Here, the lines between object/class icons do not represent association or aggregation. Rather, they represent dependencies. This is a non-standard use of the O/C model but may be useful in replacement of the matrix style of presentation. Notice that there is only one extra

	Borrower	Borrower Card	Fine	Librarian	Library Item	Overdue List
Borrower	X	X	X	-	X	-
Borrower Card	X	X	-	-	-	-
Fine	X	-	X	X	-	-
Librarian	-	-	X	X	X	X
Library Item	X	-	-	X	X	X
Overdue List	-	-	-	X	X	X

Table 7: Transitive closure matrix

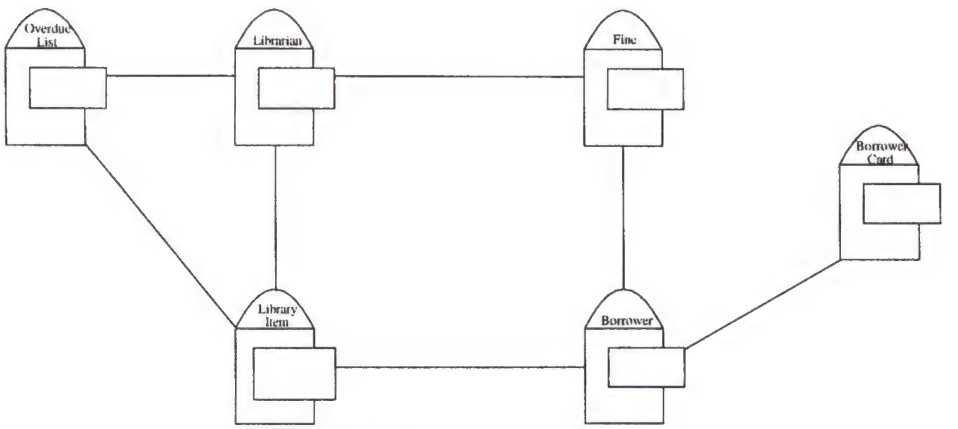


Figure 11: Transitive closure in MOSES O/C notation

"edge" added to the graph, as the transitive closure is very similar to the original graph in this example.

The dependency graph or matrix is used to show which objects/classes may operate concurrently, as they are independent of each other. A is considered independent of B if A uses none of B's services and does not use services of other objects that use B's services. The "-" symbols in the dependency matrix are replaced with "a" symbols, to represent concurrency due to independence which is termed A grade concurrency (Table 8).

Examination of Figure 11 and Table 8 suggests that the following concurrent relationships may exist in a software implementation of the object model:

- Borrower with {Librarian, Overdue List}
- Borrower Card with {Fine, Librarian, Library Item, Overdue List}
- Fine with {Borrower Card, Library Item, Overdue List}
- Librarian with {Borrower, Borrower Card)
- Library Item with {Borrower Card, Fine}
- Overdue List with {Borrower, Borrower Card, Fine}

	Borrower	Borrower Card	Fine	Librarian	Library Item	Overdue List
Borrower	X	X	X	a	X	a
Borrower Card	X	X	a	a	a	a
Fine	X	a	X	X	a	a
Librarian	a	a	X	X	X	X
Library Item	X	a	a	X	X	X
Overdue List	a	a	a	X	X	X

Table 8: The potential concurrency matrix due to object/class independence

Two refinements may be made to the transitive closure (dependency) method for identifying potential concurrency. The first of these deals with asynchronous communications. The second deals a more rigorous investigation of dependencies. Each will be discussed briefly.

3.3.1 Asynchronous Communications

Object/class dependencies were identified where one object/class provided a service to another (or a service provider of another). This was considered to prevent concurrent operation of those objects/classes. However, if the objects/classes communicate using asynchronous communication, concurrency is still possible. An examination of event model diagrams where two objects/classes in question interact, would show where they interact asynchronously.

If all interaction between two objects/classes is via asynchronous communication, then the potential for concurrency is good since the client object/class can continue processing after it sends a message requesting a service to the server object/class. This form of concurrency is represented with a "b", for B grade concurrency.

Figures 5 to 8 from the library example may be examined to see how this would affect the identification of potentially concurrent objects. If each "X" is examined from Table 8 and the corresponding objects/classes examined in each event model diagram, several object/class pairs communicate asynchronously. Table 9 shows the adjusted matrix. Figure 12 provides the adapted O/C model view of potential concurrency, introduced in Figure 11. Solid lines between objects/classes represent dependencies while dotted lines represent B grade concurrency.

	Borrower	Borrower Card	Fine	Librarian	Library Item	Overdue List
Borrower	X	X	b	a	b	a
Borrower Card	X	X	a	a	a	a
Fine	b	a	X	b	a	a
Librarian	a	a	b	X	X	X
Library Item	b	a	a	X	X	X
Overdue List	a	a	a	X	X	X

Table 9: The potential concurrency matriux, after asynchronous communication has been considered

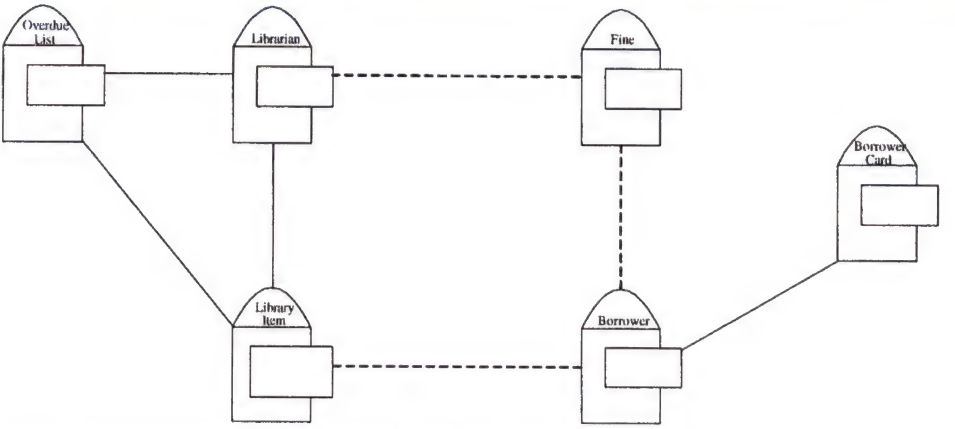


Figure 12: Dependencies between objects/classes after consideration of asynchronous communications

3.3.2 Dependency Investigation

The proposed technique identifies potential concurrency where there is no interference between objects (type A concurrency) or there is only asynchronous communications between objects (type B concurrency). There may still be the ability for two objects initially marked as not concurrent ("X" on the transitive closure matrix) to execute concurrently most of the time. Examination of event models in very fine detail would allow the identification of these situations.

- Concurrency is possible if the service provider is shared. For example, in Figure 13, object A shares B with object C. If A and C can operate concurrently, this means that B may provide a service to C while A is doing something else. This potential for concurrency is labeled with a "c", for C grade concurrency. Object transactions or concurrency mechanisms will be needed if A, B and C are allocated to different virtual nodes.

  In the library example, consideration of object sharing does not provide any new concurrency potential as grade A concurrency exists between Borrower and Librarian which share Fine, and Overdue List and Borrower which share Library Item. Therefore, the concurrency matrix remains unchanged from Table 9.

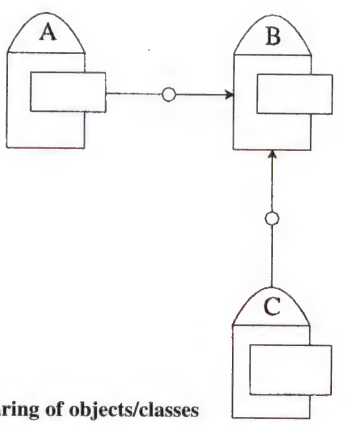


Figure 13: Sharing of objects/classes

- According to the transitive closure method, object/class A is dependent on object/class C in Figure 14. This is because A requires the services of B and B requires the services of C. In some cases, A may not require the services of C. One possible reason may be that B requires the services of C to implement services that it provides to D but is able to provide its services to A without C. Where this occurs modify the entry in the potential concurrency matrix (Table 9) to indicate concurrency between A and C. In effect it is "A" grade concurrency.

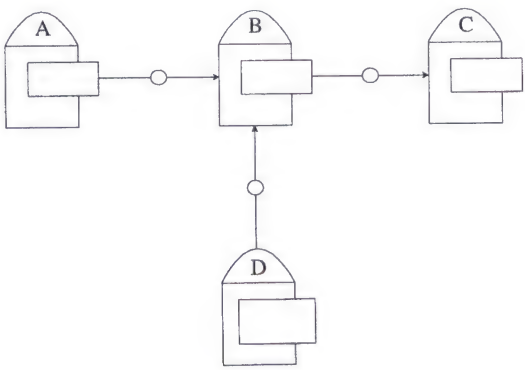


Figure 14: B's interaction with C may not be required to implement the services it provides to A

3.3.3 Summary of Steps to Identify Potential Concurrency

In summary, the following steps should be taken to identify potential concurrency in an object model:

1. Build a matrix where the rows represent objects/classes and columns represent the objects/classes that require services from this object. A "X" is shown where this is the case. In addition set the diagonal entries in the matrix to "X".
2. Find the transitive closure of the matrix using Warshall's algorithm.
3. The matrix and corresponding O/C model graph now represents the inter-O/C dependencies that prevent concurrency between objects/classes. Set all "-" symbols to "a" as these objects do not collaborate.
4. Replace "X"s with "b" if there is asynchronous communications between objects/classes. This requires an examination of all event model diagrams where those objects/classes interact.
5. Consider the dependency relationships more rigorously (optional). In particular, look for object/class dependencies that appear in the O/C model diagrams (these provide dependencies at a coarse level of granularity), but are not existent in the event model diagrams (which provide dependencies at a fine level of granularity). Failure to perform this step may reduce the opportunity for concurrency.

3.4 Size of software components

If object persistence is possible, then the size of a component is the aggregate of:

– the size of its executable image.
  Executable image size may be estimated using the number of classes, the number of methods from all classes, or the number of lines of code from all classes. While the number of classes is a very rough estimate of size, the number of methods from classes may not be - this is due to

the small number of lines that are typically found in the methods, leaving little room for variation from the average.

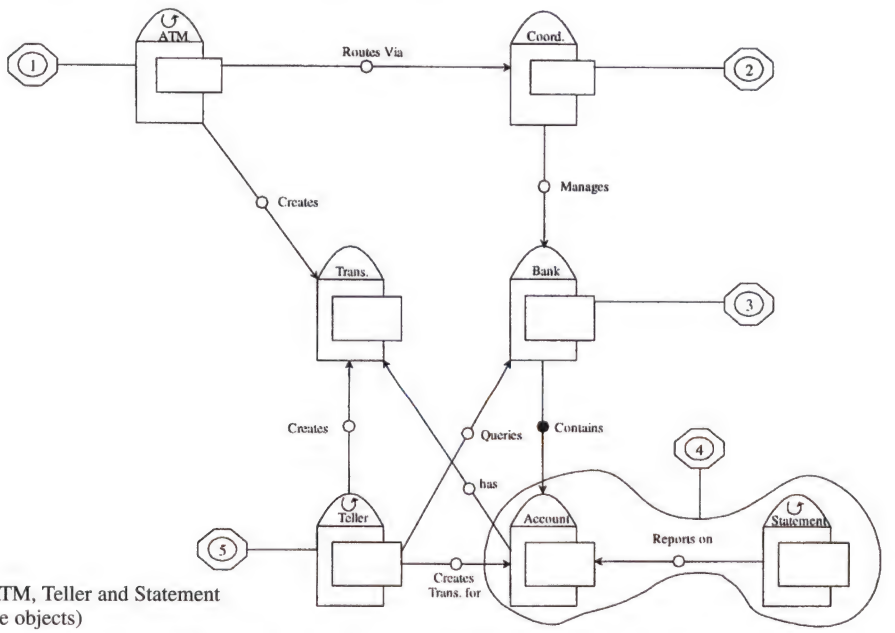
- the aggregate size of all memory resident objects.
  The memory required for objects at run time may be calculated by aggregating the size of their instance variables. In the estimation of component size, size calculations are applied recursively to all private objects. Objects that are referred to but not held privately, should be included as object references in the size count, as they will be counted in their own right.
- the aggregate size of one of each of its persistent objects.
  If no persistence is used, then all instances must be in memory. In this case, the size of the component is the aggregate of the first two points above.

4. ALLOCATION

Section 4.1 presents an example system used to illustrate the allocation process. Sections 4.2 and 4.3 provide guidelines for the derivation of information on the communications between objects and the component execution costs. The aim of this section is to use the object-oriented model to produce information that fits into non object-oriented decision techniques.

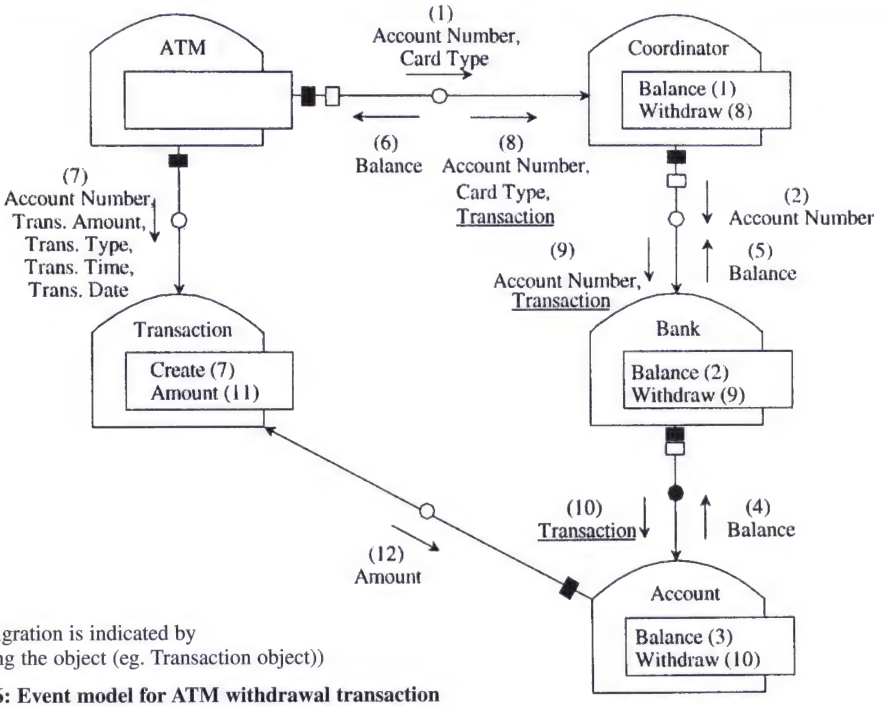
4.1 Example system

Consider the O/C model shown in Figure 15. This is the object model for a bank transaction processing system, whose requirements are shown in the side bar below. For the sake of this example, it is assumed that the events in Figures 16 to 19 are the only events that may occur in this O/C model. Tables 10 to 13 show the event traces for those events. The communication costs are based on the standard cost table shown in Table 14 and the source code for the transaction class, which migrates within some of the events, shown in Figure 20.



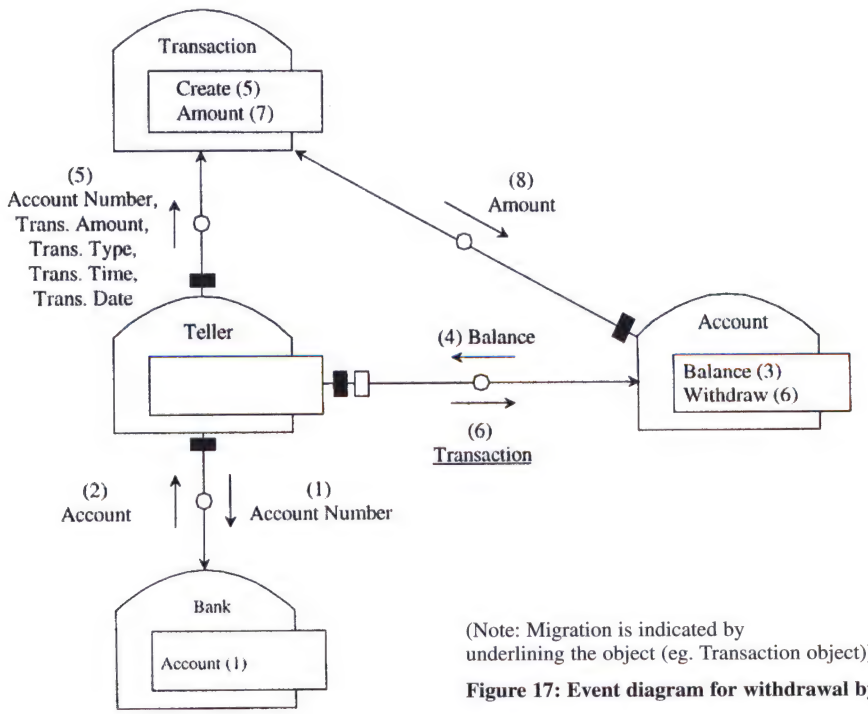
(Note: ATM, Teller and Statement are active objects)

Figure 15: O/C model with partitioning done for the bank transaction processing system



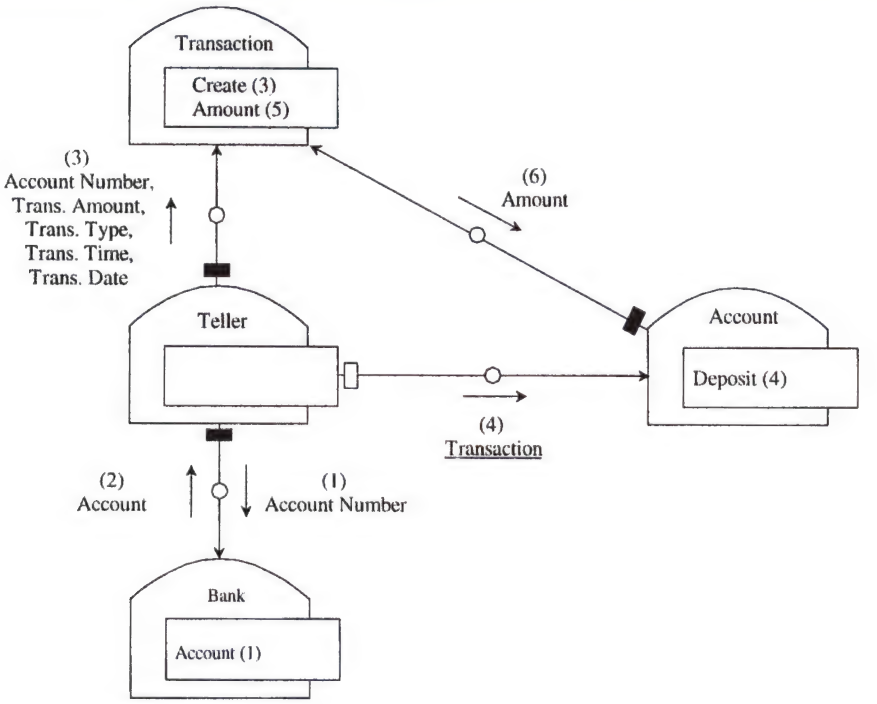
(Note: Migration is indicated by underlining the object (eg. Transaction object))

Figure 16: Event model for ATM withdrawal transaction



(Note: Migration is indicated by underlining the object (eg. Transaction object))

Figure 17: Event diagram for withdrawal by the Teller



(Note: Migration is indicated by
underlining the object (eg. Transaction object))

Figure 18: Event diagram for deposit by the Teller

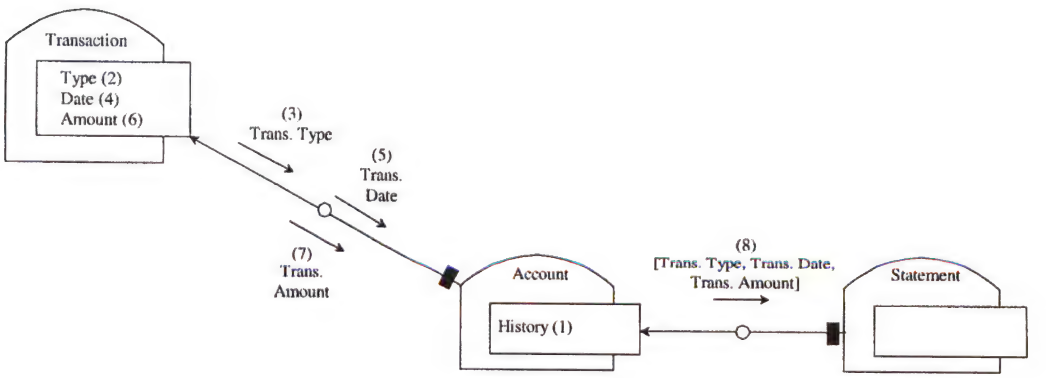


Figure 19: Event diagram for Creation of Statements

ATM Withdrawal			*10,000 per day*								
Msg	**From**	**Virtual Node**	**To**	**Virtual Node**	**Service Requested**	**Arg.**	**Arg. Type**	**Ref, Copy, Deep Copy**	**Instances**	**Arg. Cost**	**Repeat**
1	ATM	1	Coordinator	2	Balance	Account No. Card Type	Integer Integer	copy copy	1 1	1 1	1
2	Coordinator	2	Bank	3	Balance	Account No.	Integer	copy	1	1	1
3	Bank	3	Account	4	Balance	-	-	-	-	-	1
4	Account	4	Bank	3	RETURN	Balance	Dollar	copy	1	1	1
5	Bank	3	Coordinator	2	RETURN	Balance	Dollar	copy	1	1	1
6	Coordinator	2	ATM	1	RETURN	Balance	Dollar	copy	1	1	1
7	ATM	1	Transaction	1	Create	Account No. Trans Amount Trans. Date Trans. Time Trans. Type	Integer Dollar Date Time Char.	copy copy copy copy copy	1 1 1 1 1	n/a n/a n/a n/a n/a	1
8	ATM	1	Coordinator	2	Withdraw	Account No. Transaction Card Type	Integer Transaction Integer	copy copy copy	1 1 1	1 6 1	1
9	Coordinator	2	Bank	3	Withdraw	Account No. Transaction	Integer Transaction	copy copy	1 1	1 6	1
10	Bank	3	Account	4	Withdraw	Transaction	Transaction	copy	1	6	1
11	Account	4	Transaction	4	Amount	-	-	-	-	n/a	1
12	Transaction	4	Account	4	RETURN	Amount	Dollar	copy	1	n/a	1

Table 10: Trace for event model in Figure 16

Teller Withdrawal				*2,000 per day*								
Msg	**From**	**Virtual Node**	**To**	**Virtual Node**	**Service Requested**	**Arg.**	**Arg. Type**	**Ref, Copy, Deep Copy**	**Instances**	**Arg. Cost**	**Repeat**
1	Teller	5	Bank	3	Account	Account No.	Integer	copy	1	1	1
2	Bank	3	Teller	5	RETURN	Account	Account	ref	1	2	1
3	Teller	5	Account	4	Balance	-	-	-	-	-	1
4	Account	4	Teller	5	RETURN	Balance	Dollar	copy	1	1	1
5	Teller	5	Transaction	5	Create	Account No. Trans Amount Trans. Date Trans. Time Trans. Type	Integer Dollar Date Time Char	copy copy copy copy copy	1 1 1 1 1	n/a n/a n/a n/a n/a	1
6	Teller	5	Account	4	Withdraw	Transaction	Transaction	copy	1	6	1
7	Account	4	Transaction	4	Amount	-	-	-	-	n/a	1
8	Transaction	4	Account	4	RETURN	Amount	Dollar	copy	1	n/a	1

Table 11: Trace for event model in Figure 17

Teller Deposit			*1,000 per day*								
Msg	**From**	**Virtual Node**	**To**	**Virtual Node**	**Service Requested**	**Arg.**	**Arg. Type**	**Ref, Copy, Deep Copy**	**Instances**	**Arg. Cost**	**Repeat**
1	Teller	5	Bank	3	Account	Account No.	Integer	copy	1	1	1
2	Bank	3	Teller	5	RETURN	Account	Account	ref	1	2	1
3	Teller	5	Transaction	5	Create	Account No. Trans Amount Trans. Date Trans. Time Trans. Type	Integer Dollar Date Time Char.	Copy copy copy copy copy	1 1 1 1 1	n/a n/a n/a n/a n/a	1
4	Teller	5	Account	4	Deposit	Transaction	Transaction	copy	1	6	1
5	Account	4	Transaction	4	Amount	-	-	-	-	n/a	1
6	Transaction	4	Account	4	RETURN	Amount	Dollar	copy	1	n/a	1

Table 12: Trace for event model in Figure 18

Create Statements			*1 per day (max)*								
Msg	**From**	**Virtual Node**	**To**	**Virtual Node**	**Service Requested**	**Arg.**	**Arg. Type**	**Ref, Copy, Deep Copy**	**Instances**	**Arg. Cost**	**Repeat**
1	Statement	4	Account	4	History	-	-	-	-	n/a	50,000
2	Account	4	Transaction	4	Type	-	-	-	-	n/a	400,000
3	Transaction	4	Account	4	RETURN	Trans. type	Char.	copy	1	n/a	400,000
4	Account	4	Transaction	4	Date	-	-	-	-	n/a	400,000
5	Transaction	4	Account	4	RETURN	Trans. date	Date	copy	1	n/a	400,000
6	Account	4	Transaction	4	Amount	-	-	-	-	n/a	400,000
7	Transaction	4	Account	4	RETURN	Trans amount	Dollar	copy	1	n/a	400,000
8	Account	4	Statement	4	RETURN	Trans. Type Trans. Date Trans amount	Char. Date Dollar	copy copy copy	8 8 8	n/a n/a n/a	50,000 50,000 50,000

Table 13: Trace for event model in Figure 19

Table 14:
Standard type cost table

Type	Cost per Instance
Service Invocation (no argument)	3
Object Reference	2
Character	2
Integer	1
Real	2
Date	2
Time	1
Dollar	1
Ordinal	1

```
class transaction
        trans_time : Time;
        trans_date : Date;
        trans_amount : Dollar;
        trans_type : {Deposit, Withdrawal}
        account_number : Integer
visible
        create (t : Type; d : Date; a : Dollar; ti : Time; ac : Integer)
                Precondition
                        TRUE
                Postcondition
                        transaction.amount = a & transaction.type = t &
                        transaction.time = ti & transaction.account = ac
                begin
                        ...
                end
        amount : Dollar
                UNDEF;
        type : Type
                UNDEF;
        date : Date
                UNDEF;
        time : Time
                UNDEF;
end
```

Figure 20: Source Code for Transaction Class

Bank Transaction Processing System

A bank system may contain many accounts. Tellers may make withdrawal and deposit transactions on those accounts and periodically, statements will be generated showing all transactions on an account over the last month.

The bank is also connected to a third-party ATM transaction processing system, that is composed of a central coordination machine and a large number of autonomous ATM machines. ATMs only allow withdrawals. They gain access to the bank account records through the coordination machine, to check the balance before allowing the withdrawal. If the balance is sufficient, the ATM creates a transaction object and send it to the bank.

The O/C model has already been partitioned (Figure 15). It would seem that maximum concurrency is desired, as the partitioning has been done at a low level, close to the object/class natural divisions. Virtual node 4 includes the Statement object/class in with the Account - an obvious decision when the amount of communication between those two objects/classes is considered (Table 13).

The event model for ATM withdrawals is shown in Figure 16. The ATM first checks the balance of the account. To do this it sends the balance request, along with the card type and account number to the coordination object. That object routes the balance request to the appropriate bank using the card type, and the bank routes it to the appropriate account, using the account number. The balance is returned via the bank and coordination object. If the balance is sufficient for the withdrawal requested by the customer, the ATM object creates a transaction object and migrates it to the coordination object, with the card type and account number. The coordination object migrates the transaction object to the bank using the card type. The bank object migrates it to the account using the account number. The account then requests the amount of the withdrawal from the transaction object so that it can update its balance. The transaction object remains so that it may be used to produce the monthly statement.

The event model diagram for teller withdrawals is shown in Figure 17. The teller object gets the account reference from the bank object, using the account number. Then it checks the account balance. If the balance is sufficient for the requested withdrawal, the transaction object is created and migrated onto the same virtual node as the account object.

The event model diagram for a teller deposit is shown in Figure 18. The teller object does not need to know the account balance. It simply creates the transaction object, and gets the account reference from the bank object using the account number. The transaction object is then sent to the virtual node that holds the accounts.

The event model diagram for creation of account statements is shown in Figure 19. The statement object requests the history service from each of the bank's account objects. Each account object retrieves the type, date and amount for each of the transactions that have been performed on it. These details are returned to the Statement object.

4.2 Communications

4.2.1 Inter-Component Communication Costs

Three possible measures for determining the communications cost are:

- counting the number of messages sent;
- counting the number of messages sent and weighting these messages based on the number of arguments sent with the message; and
- counting the number of messages sent and weighting these messages based on the size of arguments sent.

Only messages between objects/classes that are allocated to different virtual nodes are counted in the allocation process. The first and most simple method was shown for partitioning (Section 3.1). The second method is a simple extension of the first. The third method is the most appropriate for allocation since it provides information on the actual communications load between nodes. This method will be expanded upon here.

4.2.2 Explicit Communications between Components

The event model diagrams should be examined to identify the interaction between objects that exist on different virtual nodes. The information required can be gathered from the event traces, which require a little more information to achieve the weighting of messages depending on their arguments. Tables 10 to 13 show the traces for the preceding event model diagrams. The shaded rows in the tables highlight those messages that are sent between different virtual nodes (inter-component messages). Those are the messages that must be costed.

The event trace tables have been extended to show the type and number of objects sent as arguments with messages. These details are combined to determine the cost of each argument. The type comes from the event model and the number of objects would generally be an estimate made on the basis of domain knowledge. The cost for each object in the argument may come from one of two places. Costs for object references and standard types (e.g. integers) are based on the programming language and distributed computer system to be used (e.g. Table 14). The cost of user defined objects that are contained in the message (migration) should be based on size estimates. These are based on the instance variables and objects within the implementation of the user defined object. To get the most accurate size estimates, the source code must be available. Less accurate estimates could be based on the number of attributes expected in the object/class implementation. If the arguments involve a deep copy, then the cost of an object in the argument should include the cost of all objects it references.

The source code for the transaction class is shown in Figure 20. This is required to determine the cost of migrating an instance of the class between virtual nodes. The cost is calculated by summing the cost of the instance variables in the class. If any of those instance variables are user defined types (i.e. instances of other classes), then two approaches may be taken. If the copy is not a deep one, then the cost of the instance variable is only the cost of an object reference, which is dependent on the distributed computer system in use. If the copy is a deep one, the cost of the instance variable is the cost of migrating the object it refers to. The cost of doing that is calculated by recursively applying the approach being described on the object(s) referred to by the instance variable.

The cost of sending a transaction object is six standard cost units (see Table 14). This would be the same if a deep copy were used as the transaction object does not refer to any other objects.

(Note: Communications costs shown have been divided by 1000)

Table 15: Matrix of inter-component communication costs (relative)

Component	1	2	3	4	5
1	∞	200	0	0	0
2	-	∞	180	0	0
3	-	-	∞	160	27
4	-	-	-	∞	41
5	-	-	-	-	∞

Table 15[2] shows the cost matrix for communications between virtual nodes using equation 1. Figure 21 shows the calculation of costs for Table 15, using the trace tables (10-13), and the standard cost table (Table 14). The communication costs may be considered relative to each other. If the standard cost table is done properly, they may also represent a volume, such as bits. Table 15 simply shows the relative costs.

Figure 22 may be obtained by removing the object/class icons in the O/C model from Figure 15. This leaves the virtual node icons behind, which can be connected to form the nodes of a graph. The level of communication between each virtual node can be shown as weights on the edges between the nodes of that graph (Figure 22). Tidying this up and removing the edges that have a zero weight results in the graph shown in Figure 23. An intelligent CASE tool might allow this as an alternative view of the O/C model diagram from Figure 15.

The graph in Figure 23 is no different to those used in the distributed literature, where each node represents a process and the weight on the edges represents the level of interprocess communication (IPC) between the two processes (nodes). Graph theoretic allocation techniques (Jenny, 1977; Stone and Bokhari, 1978) and some heuristic allocation techniques (Gylys and Edwards, 1976; Efe, 1982) may then take this as input. They also require information on the amount of processing required in each process (virtual node in this paper), which is discussed in section 4.3.

4.2.3 Implicit Communications between Components

The method described above, works on the basis that the only inter-component communications are shown in the event model diagrams. Clearly, this is dependent on the existence of event model diagrams for each system scenario. However, event model diagrams, even if they do cover all scenarios, may not capture all communication between components. One reason for this would be replicated objects. These must remain consistent, which involves inter-component communication of messages, outside of the event models. Another reason may be the concurrency controlling mechanism provided by the distributed computer system being used. Yet another may be the cost of binding remote objects to their messages, which may involve a name server elsewhere in the system.

[2] Communictions costs shown are divided by 1000.

Calculation of Inter-Component Communication Costs

Between virtual nodes 1 and 2, cost =

50,000	(fig. 16, msg. 1)
40,000	(fig. 16, msg. 6)
110,000	(fig. 16, msg. 8)
200,000	

Between virtual nodes 2 and 3, cost =

40,000	(fig. 16, msg. 2)
40,000	(fig. 16, msg. 5)
100,000	(fig. 16, msg. 9)
180,000	

Between virtual nodes 3 and 4, cost =

30,000	(fig. 16, msg. 3)
40,000	(fig. 16, msg. 4)
90,000	(fig. 16, msg. 10)
160,000	

Between virtual nodes 4 and 5, cost =

6,000	(fig. 17, msg. 3)
8,000	(fig. 17, msg. 4)
18,000	(fig. 17, msg. 6)
9,000	(fig. 18, msg. 4)
41,000	

Between virtual nodes 3 and 5, cost =

8,000	(fig. 17, msg. 1)
10,000	(fig. 17, msg. 2)
4,000	(fig. 18, msg. 1)
5,000	(fig. 18, msg. 2)
27,000	

Figure 21: Calculation of inter-component communication costs

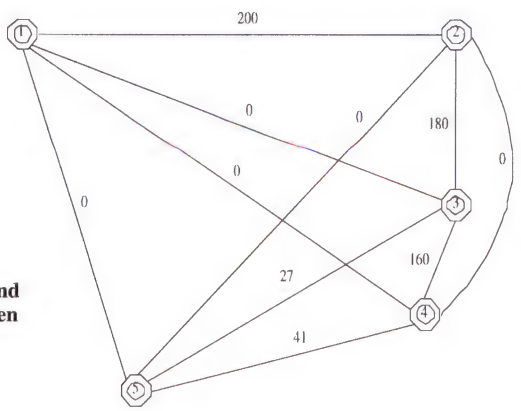


Figure 22: O/C model with object/class icons removed and weighted edges added between the virtual nodes

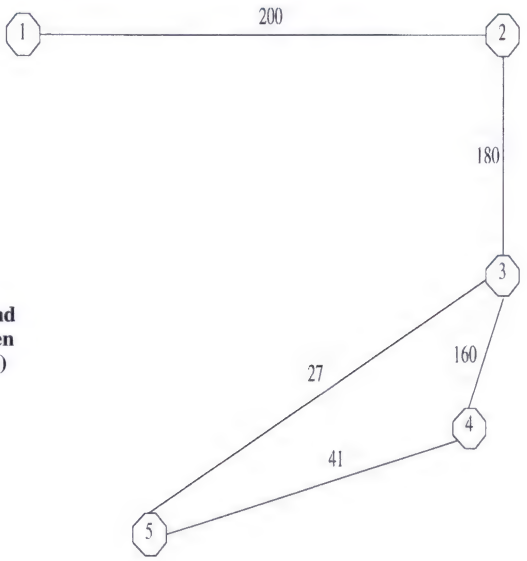


Figure 23: O/C model with object/class icons removed and weighted edges added between the virtual nodes (cleaned up)

Replication

The cost of replication may be captured by making assumptions about how often the objects update each other. If they maintain themselves in synchronisation at all times, any update message that arrives at one object will be sent to all its replicates. Therefore, whenever an object that is replicated receives a message, the cost of that message should be applied to each link between the first object's virtual node and the virtual nodes where the replicates reside.

Concurrency Controlling Mechanism

When an object is shared by objects in different threads of control (i.e. on different virtual nodes), concurrent access to that object must be controlled. There are various methods to do this. Firstly there are concurrent programming techniques such as use of semaphores and monitors (Ben-Ari, 1990; Martin *et al,* 1991). Secondly, there are correctness criteria. An example of a correctness criterion approach is to use two phase locking which ensures serializability (Martin *et al,* 1991; Desai, 1990). Other correctness criterion approaches might also aim for serializability but through the use of versions or timestamps (Desai, 1990).

Which of these mechanisms (monitors, semaphores, locking, timestamps, versions) is used, depends on the distributed computer system being used to support the application. Some of the mechanisms involve more messages from the client object to the server than the simple interaction model presented till now. For example, locking requires a message requesting the transaction with a return message when the transaction is accepted. Semaphores also require an extra message which blocks the client object until the server object sends a signal to the semaphore indicating that it is ready to accept another service request. The client may then send its actual message. Both of these concurrency control mechanisms require two additional messages. The locking mechanism may only require the messages between two objects once in an event, even if the objects send several messages to each other. Furthermore, the distributed computer system may support locking of many objects at once (perhaps all instances of a class), in which case the transaction request would not be required for each object-object interaction (shown by the number

in the Repeat column of the event trace). On the other hand, the semaphore mechanism may require the extra two messages each time a message is sent within an event. These details should be added to the event trace table.

Object Binding

Object binding may require messages to name servers that convert an object name to a physical address on the network. This would be an attribute of the distributed computer system employed to support the application. The name server might be represented as a virtual node, and the communications to that virtual node included for each object message sent. Whether this is required or not would depend on the way the distributed computer system handles binding.

4.3 Component Execution Costs

The execution cost of having a virtual node on a processor is difficult to estimate. The number of lines of code that a virtual node would require to be executed in a period of interest gives an approximate surrogate measure of the execution cost. Obviously the cost of executing a line of code is not constant. However if there is a large number of lines of code, these variations should average out.

The trace tables used for calculating communication costs are invaluable here. Each service request will involve the execution of a number of lines of code. Therefore, the number of lines of code required by a virtual node in some period of interest is equal to the number of lines of code in the services requested of objects allocated to that virtual node within that period. The formula for this calculation is presented in equation (1).

$$Cost_x = \sum_{\forall i; i \in E} (n_i \times \sum_{\forall j,k \leftrightarrow l,k \in x} (m_j \times loc_{kj})) \qquad (1)$$

where:

x	=	virtual node
$Cost_x$	=	component execution cost for virtual node x
E	=	set of all events
$\forall i; i \in E$	=	all events (i) from the set of events E
n_i	=	the number of times event i is repeated in a period of interest
$\forall j; k \leftrightarrow l, k \in x$	=	all messages (j) where l requests a service from k and k is on virtual node x (i.e. does not include RETURN messages)
m_j	=	the number of times the message (j) is sent between k and l in one occurrence of event i
loc_{kj}	=	the number of lines of code needed to implement the service provided by k, requested by l by message j in event i

These costs may then be shown on the nodes as done in Figure 24 for the bank transaction processing system example. The source code for each method is required to achieve this but is not shown here. The calculations required to find the execution costs shown in Figure 24 appear in Figure 25. They are based on an estimate of the number of lines for code for each service called in the event models shown in Figures 16 to 19.

Active objects are the source of chains of service requests - these objects contain private services and LIVE routines that must be included in the count. They are not the result of service

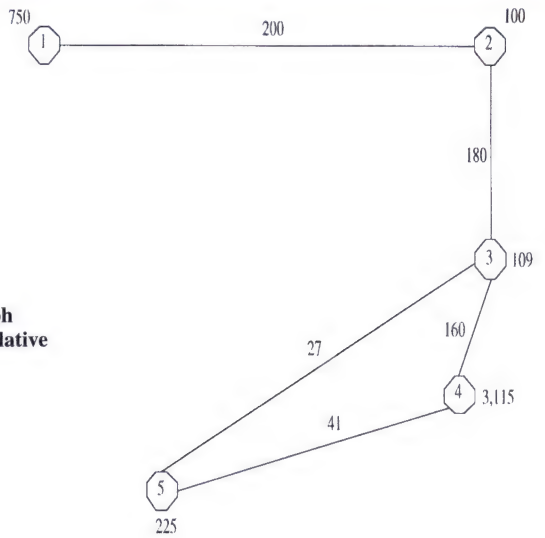


Figure 24: Virtual node graph with communications and relative execution costs shown

requests shown in the event trace, and may be forgotten if care is not taken. Where the LIVE routine is a constant loop, it is very difficult to say what the execution cost will be. One technique would be to estimate the number of times the routine would execute in the period of interest and then use equation 1.

Once the number of lines of code that each virtual node will need executed in a period of interest is calculated, the various speeds of physical nodes and types of code may be considered. This will mean that two virtual nodes with x lines of code may have different execution costs depending on the type of code (language and characteristics such as I/O intensity). It also means that two physical nodes may have different execution costs for the same virtual nodes. These considerations are dealt with in the graph theoretic allocation methods and are not discussed further in this paper.

4.4 Bringing Communication and Execution Costs Together

Constraints such as required and disallowed allocations may also be introduced at this point. Again, treatment of issues such as these are not unique to object-orientation and are not discussed further.

Figure 26 shows the introduction of physical nodes to the bank transaction processing system example. There are five physical processors (A,B,C, D and M). The ATM virtual node must be allocated to the M processor (ie. it is an ATM machine). It has been decided that the coordinator function will be allocated to the A processor. It has been further decided that the other virtual nodes may not be allocated to either processor A or M. These constraints are shown by putting infinite execution costs on the links between the virtual node and the physical node.

Figure 27 removes the lines between virtual nodes and physical nodes where allocations are disallowed to ease understandability. The letters in brackets after the execution cost figures represent weightings to allow for the differing execution speeds of the physical nodes. Figure 27 may now be used by graph theoretic allocation techniques (Jenny, 1977; Stone and Bokhari, 1978) and some heuristic allocation techniques (Gylys and Edwards, 1976; Efe, 1982).

Calculation of Component Execution Costs

Virtual node 1 : cost =

```
600,000(secret services in ATM - fig. 16)
150,000(fig. 16, msg. 7)
----------
750,000
```

Virtual node 2 : cost =

```
 50,000(fig. 16, msg. 1)
 50,000(fig. 16, msg. 8)
----------
100,000
```

Virtual node 3 : cost =

```
 50,000(fig. 16, msg. 2)
 50,000(fig. 16, msg. 9)
  6,000(fig. 17, msg. 1)
  3,000(fig. 18, msg. 1)
----------
109,000
```

Virtual node 4 : cost =

```
 20,000(fig. 16, msg. 3)
 50,000(fig. 16, msg. 10)
 20,000(fig. 16, msg. 11)
  4,000(fig. 17, msg. 3)
 10,000(fig. 17, msg. 6)
  4,000(fig. 17, msg. 7)
  5,000(fig. 18, msg. 4)
  2,000(fig. 18, msg. 5)
100,000(secret services in Statement - fig. 19)
500,000(fig. 19, msg. 1)
800,000(fig. 19, msg. 2)
800,000(fig. 19, msg. 4)
800,000(fig. 19, msg. 6)
----------
3,115,000
```

Virtual node 5 : cost =

```
120,000(secret services in teller - fig. 17)
 30,000(fig. 17, msg. 5)
 60,000(secret services in teller - fig. 18)
 15,000              (fig. 18, msg. 3)
----------
225,000
```

Figure 25: Calculation of component execution costs

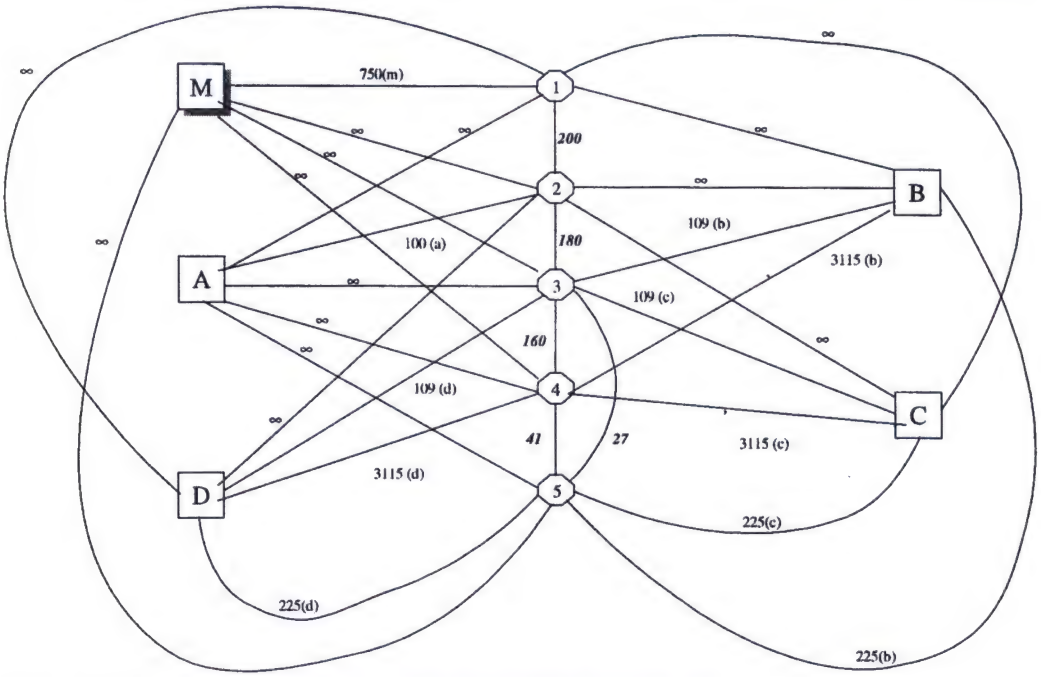


Figure 26: Allocation graph with constraints shown with infinite execution costs

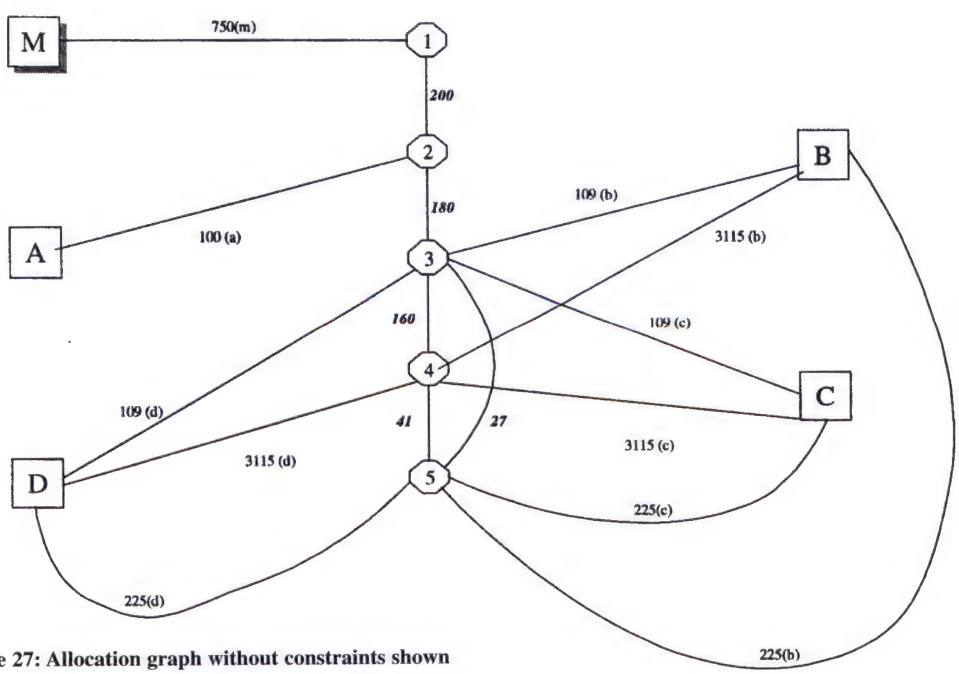


Figure 27: Allocation graph without constraints shown

5. CONCLUSIONS

The extended MOSES notation for distributed systems has been used to support software partitioning and allocation. Techniques for evaluating the three objectives of task partitioning in object oriented development have been specified: minimization of intermodule communications, exploitation of concurrency, and limiting the size of processes. The potential concurrency in the object model, if identified as described in Section 3.3, is graded into three categories. This grading of the potential concurrency between objects/classes may assist in the partitioning decision making process. It is not included in current partitioning techniques (Shatz and Yau, 1986) and therefore requires more investigation. The resultant processes are allocated to virtual nodes using standard non object-oriented decision techniques.

The linking of the virtual nodes to physical nodes occurs in the allocation process which considers issues such as inter-virtual node communication costs and virtual node execution costs. Inter-virtual node communication is calculated from the event diagrams and event traces, weighting messages by the size and number of arguments they include. The mathematical working involved in this could easily be placed in spreadsheet macros. Intelligent CASE tools could go further than that, and obtain information straight from the event model diagrams.

Execution costs for each virtual node are also calculated using the event traces and the number of lines of code in each method that is called while its object is allocated to the virtual node.

The outputs of the allocation process described in this paper (e.g. Figure 27) can go straight into traditional graph theoretic or graph heuristic allocation techniques.

Current work includes extending the UML model to provide improved support for the consideration of distribution issues and continued validation of the techniques presented in this paper. Ongoing research is also considering other task allocation cost metrics.

REFERENCES

ATKINSON, C. (1991): *Object-Oriented Reuse, Concurrency and Distribution: An Ada-Based Approach,* ACM Press, 270

BANNISTER, J.A. and TRIVEDI, K.S. (1983): Task Allocation in Fault-Tolerant Distributed Systems *Acta Informatica,* 20: 261-281

BEN-ARI, M. (1990): *Principles of Concurrent and Distributed Programming,* Prentice-Hall, 225

BENNETT, J.K. (1987): The Design and Implementation of Distributed Smalltalk *OOPSLA '87 Proceedings,* October 4-8: 318-330

BLACK, A., HUTCHINSON, N., JUL, E., LEVY, H. and CARTER, L. (1987): Distribution and Abstract Types in Emerald, *IEEE Transactions on Software Engineering,* 13: 1, 65-76

CHU, W., HOLLOWAY, L., LAN, M., and EFE, K. (1980): Task Allocation In Distributed Data Processing, *IEEE Computer,* November: 57-69

DESAI, B.C. (1990): *An Introduction to Database Systems,* St. Paul, MN: West Publishing Company, 820

EFE, K. (1982): Heuristic Models of Task Assignment Scheduling in Distributed Systems, *IEEE Computer,* 15(6): 50-56

EZZAT, A.K., BERGERON, R.D. and POKOSKI, J.L. (1986): Task Allocation Heuristics for Distributed Computing Systems, *Proceedings of the 6th International Conference on Distributed Computer Systems,* Washington, IEEE Computer Society Press: 337-345

FLOYD, R.W. (1962): "Algorithm 97 (SHORTEST PATH)", *Communications of the ACM,* 5(6): 345

GYLYS, V.B. and EDWARDS, J.A. (1976): Optimal Partitioning of Workload for Distributed Systems, *Proceedings of Compcon,* Fall 353-357

HARIRI, S. and RAGHAVENDRA, C.S.(1986): Distributed Functions Allocation for Reliability and Delay Optimization, *Proceedings of the 1986 Fall Joint Computer Conference,* 344-352

JENNY, C.J. (1977): Process Partitioning in Distributed Systems, *Digest of Papers National Telecommunications Conference '77,* 31:1-1 to 31:1-10.

LEE, R.P. and MUNTZ, R.R. (1977): On the Task Assignment Problem for Computer Networks, *Proceedings 10th Hawaii International Conference on System Sciences,* January: 5-9

LOW, G.C., RASMUSSEN, G. and HENDERSON-SELLERS, B. (1996): Incorporation of Distributed Computing Concerns into Object-Oriented Methodologies, *J.O.O.P,* 9(3): 12-20

MARTIN, B.E., PEDERSEN, C.H. and BEDFORD-ROBERTS, J. (1991): An Object-Based Taxonomy for Distributed Computer Systems, *IEEE Computer,* August: 17-27

MÜHLHAÜSER, M., W. GERTEIS and L. HEUSER (1993): DOCASE: A Methodic Approach to Distributed Programming, *Communications of the ACM,* 36(9): 127-138

RASMUSSEN, G., HENDERSON-SELLERS, B. and LOW, G.C. (1996a): Extending the MOSES Methodology to Distributed Systems , *J.O.O.P.,* 9(4): 39-46

RASMUSSEN, G., HENDERSON-SELLERS, B. and LOW, G.C. (1996b): An Object-Oriented Analysis and Design Notation for Distributed Systems, *J.O.O.P.,* 9(6): 14-27

SHATZ, S.M. (1993): *Development of Distributed Software: Concepts and Tools,* Macmillan Publishing Company, 209

SHATZ, S.M. and Yau S.S. (1986): A Partitioning Algorithm for Distributed Software Systems Design, *Information Sciences,* 38: 165-180

SHATZ, S.M. and WANG, J. (editors) (1989a): *Tutorial: Distributed-Software Engineering,* IEEE Computer Society, Washington DC: 279

SHATZ, S.M. and WANG, J. (1986B): Models & Algorithms for Reliability-Oriented Task-Allocation in Redundant Distributed-Computer Systems, *IEEE Transactions on Reliability,* 38(1): 16-27

STONE, H.S. and BOKARI, S.H. (1978): Control of Distributed Processes, *Computer,* 11(7): 97-106

BIOGRAPHICAL NOTES

Graham Low is an Associate Professor and Head of the School of Information Systems, Technology and Management at the University of New South Wales. Graham's research has focused on software engineering with a particular interest in distributed application development and its enabling technologies such as object-oriented development and networking. The conduct of empirical studies in industry is an important part of this programme. Prior to joining the University in 1985, Graham was Technical MIS Manager for the Suger Division of CSR.

Geoff Rasmussen was educated at UNSW receiving a BSc (BIT) and the University Medal (shared). He is now a director of consulting and corporate advisory firm Poynton and Partners and is based in Perth, Western Australia. Prior to founding Poynton and Partners, Geoff was with McKinsey and Company. He is a board member of the listed internet software company HarvestRoad Limited and the Electronic Commerce Network, part of Curtin University Business School.

Applying the ACS Code of Ethics

Oliver K. Burmeister

Lecturer
School of Information Technology
Swinburne University of Technology
and Executive member, Australian Institute of Computer Ethics
Email: oliver@it.swin.edu.au

The IT professional unlike professionals in other disciplines does not have to abide by the strictures of a professional society. In Australia the professional IT association has a Code of Ethics that, while easily accessible, needs clarification to apply it in the real world. Though the ACS code is distinctly Australian in the way it has been formulated, it sits easily within the general tenets espoused by similar associations in other countries. Although cultural issues have influenced the moral philosophy of the ACS code, there are lessons from other countries that apply in the Australian context. Interpreting the code and applying it to one's situation can be facilitated through seeing how others have applied the code and through understanding its underlying tenets.

Key Words: code, clients, employers, ethics, profession, public

1. INTRODUCTION

This paper begins with a discussion of what a professional code of conduct and in particular a code of ethics aims to achieve. It focuses in particular on how professionals in Australia have interpreted the Australian Computer Society (ACS) Code of Ethics (code) in its various forms over more than twenty years. Next the major tenets of the ACS code are examined in the light of its historical development and in the light of similar tenets by two similar organisations, the Association for Computing Machinery (ACM) and of the British Computer Society (BCS). The paper then restates the ACS code in its current form, before going on to apply that code to nine case studies. These case studies have been used in a similar context by other IT professionals as a reasonable representative sample of likely scenarios IT practitioners may encounter.

2. WHY CODIFY ETHICS?

There are numerous instances in one's professional life that call for judgement not so much in technical matters, as in matters of professional efficacy. The Australian Computer Society (ACS) has sought to help members and the wider professional body in Australia by codifying the minimal acceptable standards for an IT professional. This Code of Ethics and its associated Standard of Conduct (together referred to as "code" throughout most of this paper) have existed for some years, yet frequently members have expressed the view that the code is not well understood, or that it does not cater well to their circumstances. The first of these issues is addressed in this paper. The second

Copyright© 2000, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the JRPIT copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

Manuscript received: August 1999
Editor: David Wilson

has been partly addressed by the society. The ACS set up an Ethics Task Force that surveyed members as to their attitudes to a number of ethics issues. The results of that survey will help in the ongoing process of keeping the ACS code relevant to current professional issues.

The code is not a fixed, immutable standard. In fact, different countries have different codes, and in each case the code has evolved over time. Each country has its own unique cultural and professional influences. Each country may also have numerous professional societies that involve IT professionals. For instance, information systems professionals might be members of a society for software engineers. Similarly, IT management consultants might be part of a professional body for managers, in preference to that country's IT association. In a rapidly changing industry such as IT, ethical thinking will understandably change (though perhaps not so rapidly as the technology). What was once considered an unacceptable intrusion on one's privacy, might now no longer be considered such.

Professional codes enable both those inside and those outside the profession (for instance the general public) to evaluate exactly what might be appropriately expected from members of the profession (Woodings, 1987; Lichtenberg, 1996). A professional code describes clearly and publicly both acceptable and unacceptable professional behaviour (Coldwell, 1987). A code can also assist people new to IT in better appreciating the range of circumstances they may encounter during their professional life (Woodings, 1987; Morrison and Forester, 1990; Anderson *et al,* 1993); for instance, the ACS was the first professional IT society in the world to include in its code reference to educating IT students [other societies have since followed this lead, for instance SEERI (1998)]. Furthermore a code allows the profession as a whole to support individuals who maintain an agreed viewpoint. Langford (1995) suggests a professional code of ethics "is analogous to the establishment of international software standards. Once established, proper compliance with such standards can be advertised and depended upon. Whoever actually supplies code, and wherever in the world they may be located, customers may then be sure of what they are buying."

Professional codes are also deemed to be helpful in resolving what can often be very personal conflicts between one's work and one's beliefs. Langford (1995) suggests there are three ways in which reference to a professional code can be helpful:

1. Codes can identify both current and future problem areas. Codes help the professional to identify and think through potential ethical issues and how to resolve them BEFORE actual problems arise.
2. Because formal professional codes have been agreed to by the entire profession (or at least by the members of that society, which in the case of the ACS is claimed to be (ACS, 1999b) over 16,000 members, which the ACS estimates is approximately 20% of Australian IT professionals), they lend support and justification for the actions of individuals faced with problems that are addressed by the code.
3. Once a breach of the code has been identified, both direct and indirect help for its resolution may come from the society itself.

Compliance with codes of practice, especially where this is voluntary, presumes professionals understand the code and how to apply it. Without a code that is generally accepted, compliance is very difficult to achieve. In other professions such as law, medicine and engineering, there are accepted codes of conduct that are enforced by the professional body. In the case of IT professionals, except in breaches of the law, self-regulation is all there is (for non-ACS members). Australian IT practitioners are not required to join the ACS, making compliance with the industry

body's code even more difficult to achieve. Indeed compliance is not the main purpose of such a code, instead it is "intended to serve as a basis for ethical decision making in the conduct of professional work" (ACM, 1999).

The code encourages greater public trust by this public commitment of IT professionals of whom the ACS is the representative body in Australia. The concept of 'public trust' is inherent in the very definition of professionalism. This can be seen in the following partial quote from the definition of a profession of the Australian Council of Professions (who have recently recognised ACS members as professionals, alongside doctors, lawyers, engineers and other professionals):

> "It is inherent in the definition of a profession that a code of ethics govern the activities of each profession. Such codes require behaviour and practice beyond the personal moral obligations of an individual.
> They define and demand high standards of behaviour in respect to the services provided to the public and in dealing with professional colleagues. Further, these codes are enforced by the profession and are acknowledged and accepted by the community." (ACS, 2000)

Many IT professionals both within the ACS and beyond are concerned about ethical issues. The ACS describes itself as "the public voice of the IT professional; the guardian of professional ethics and standards in IT; with a commitment to the wider community to ensure the beneficial use of IT." (ACS, 1999b) The ACS encourages voluntary self-regulation (ACS, 1988a). To achieve this practitioners need to understand what the code is and what its underlying principles are. It is also helpful to see how the code might be applied in real cases.

3. PRINCIPLES OF THE ACS CODE

What are the underlying principles of the ACS code and of other similar codes? Barroso Asenjo (1997) reported on a study of fifteen codes of ethics of computing companies, conducted in 1994 and 1995, which verified earlier studies of computer society codes that revealed four main principles: privacy, accuracy (as in faithfulness and precision in data transmission), property and accessibility. The codes of the ACS, ACM and BCS all reflect similar values: as one might expect, they have not developed in a vacuum, but are improvements of earlier versions of the codes of these societies and have been built on the work of other professional bodies before them. Essentially they each exhibit what might be referred to as "Level 1" and "Level 2" statements, built around major principles. In the case of the ACS the Level 1 statements are the Code of Ethics and the Level 2 statements are the Standard of Conduct. The latter seeks to flesh out the Code of Ethics with some practical examples, aimed at helping professionals apply this idealism.

This distinction between levels began with the first version of the ACS code in 1975, when the ACS published its Code of Professional Conduct (Montgomery, 1987) which was split into the Code of Ethics and the Code of Good Practice. Major revisions in 1980 and again in 1985 have kept this structure.

The Level 1 statements are an ideal Code of Ethics that in general terms describe the responsibilities of IT professionals to the public, employers, employees, clients and other members of the IT community. Table 1 shows the headings used to encapsulate level 1 statements by each of the three societies.

ACS	BCS	ACM
Priorities	The Public Interest	General Moral Imperatives
Competence	Duty to Employers and Clients	More Specific Professional Responsibilities
Honesty	Duty to the Profession	Organisational Leadership Imperatives
Social Implications	Professional Competence and Integrity	Compliance with the Code
Professional Development		
Computing Profession		

Table 1: Guiding principles in the IT profession [1]

At first glance they appear quite different, but they address very similar issues confronting IT professionals throughout the world. The ordering is slightly different, though each one ends with compliance/professionalism related principles. Regardless of the headings, these principles represent the spirit of the code, whereas the Level 2 statements flesh out the letter of the code as the principles are currently being interpreted by the society and its members. Yet when Levels 2 statements conflict, the IT professional needs to understand the spirit behind them as a guide to ethical decision making.

All three societies are concerned to protect fundamental human rights and minimise negative consequences of computing implementations, particularly ones that threaten the health and safety of anyone involved. The main principles of all three societies ascribe to the need to consider the implications of one's work; this is integrated into the very concept of being an IT professional. "The effect of the Code of Ethics is to create for every professional member not just the right, but the duty, to regard the implications of their work as part and parcel of their application of IT." (Clarke, 1990)

3.1 The ACS Code of Ethics

The ACS has published its Code of Ethics on numerous occasions and it is most easily accessible to the public on their web site (ACS, 1999). The code is introduced with a general statement of purpose that is also a compliance requirement:

> *An essential characteristic of a profession is the need for its members to abide by a Code of Ethics. The Society requires its members to subscribe to a set of values and ideals which uphold and advance the honour, dignity and effectiveness of the profession of information technology. The member is required at all times to be honest, forthright and impartial, to loyally serve employers, clients and the public, to strive to increase the competence and prestige of the profession and to use special knowledge and skill for the advancement of human welfare.*

[1] The URLs of the codes of ethics of all three societies are given in the references of this paper.

The code:

> *I will act with professional responsibility and integrity in my dealings with clients, employers, employees, students and the community generally. By this I mean:*
>
> 1. *I will serve the interests of my clients and employers, my employees and students, and the community generally, as matters of no less priority than the interests of myself or my colleagues.*
> 2. *I will work competently and diligently for my clients and employers .*
> 3. *I will be honest in my representations of skills, knowledge, services and products.*
> 4. *I will strive to enhance the quality of life of those affected by my work.*
> 5. *I will enhance my own professional development, and that of my colleagues, employees and students.*
> 6. *I will enhance the integrity of the Computing Profession and the respect of its members for each other.*

The ACS Code of Ethics has existed in this form with only minor changes for more than a decade. The last major revision was in 1985 in response to the spread of computer based crime (Coldwell, 1987). The goal of the last revision was to promote better public awareness of the fact that honesty and integrity are valued by members of the IT profession. A significant aspect of the code is the obligation of an IT practitioner to public service and public interest (Earle and Fitzgerald, 1983).

3.2 The ACS Standard of Conduct

One of the major limitations of most professional codes (Martin and Schinzinger, 1995) is that they cannot be applied in a straightforward manner to many situations. For this reason the ACS has produced its Standard of Conduct, which spells out in level 2 statements how the Code of Ethics should be interpreted. These statements elaborate upon Code of Ethics, which is seen as "ideal, and may not all be achievable at all times in all circumstances" (ACS, 1999). As is often the case with moral considerations, there will be times when values conflict with each other. It is then that the IT professional has to weigh up "the relevant factors and choose to act in the manner which is most consistent with the Code of Ethics, given the circumstances." The ACS has not provided guidance as to which tenets have priorities over others in situations of conflict. This is where professional judgement is needed.

The Standard of Conduct restates the tenets and the Code of Ethics (one statement at a time) and gives practical guidance (see Appendix).

Now that we have examined the need for a professional code, described the ACS principles in the light of similar principles of other codes and shown the current state of the ACS Code of Ethics and Standard of Conduct, we can examine realistic cases to see how the code might be applied to them.

4. APPLYING THE CODE

How can professionals be helped to understand how to apply the code in practice? An excellent source of help for ACM members is a much quoted paper by Anderson, Johnson, Gotterbarn and Perrolle (1993). Given the wide appeal of the approach taken Anderson et al. (1993), which examines the ACM code in the light of nine realistic cases, I have chosen to use the same nine

cases and apply the ACS code to them. To avoid repeated references to the same work, each of the cases has been indented and put in italics; in a few instances cases have been slightly modified either to modernise them or to make them culturally relevant to an Australian readership.[2]

4.1 INTELLECTUAL PROPERTY

> *Jean, a statistical database programmer, is trying to write a large statistical program needed by her company. Programmers in this company are encouraged to write about their work and to publish their algorithms in professional journals. After months of tedious programming, Jean has found herself stuck on several parts of the program. Her manager, not recognising the complexity of the problem, wants the job completed within the next few days. Not knowing how to solve the problems, Jean remembers that a co-worker had given her source listings from his current work and from an early version of a commercial software package developed at another company. On studying these programs, she sees two areas of code which could be directly incorporated into her own program. She uses segments of code from both her co-worker and the commercial software, but does not tell anyone or mention it in the documentation. She completes the project and turns it in a day ahead of time.*

The third principle "I will be honest in my representations of skills ..." and in particular principles 3.2 about not misrepresenting one's skills and 3.6 about giving due credit to others, have been violated by Jean. By not giving credit to her co-worker and the commercial software that presumably was copyrighted/protected by law, Jean has violated professional ethics. She has also violated principle 1.3 about respecting the proprietary nature of other's information. Jean should have checked to see if her company was permitted to use the source code, prior using it, otherwise she might expose her employer to legal liability. Even if Jean had merely sought the source code for ideas and then completely written her own program, she should have acknowledged her co-worker in the documentation. Obviously judgement is called for here – if the intellectual contribution from the co-worker was of a trivial nature, then there would not have been a need to acknowledge it.

For further perspectives in this area, the following would be helpful: Cifuentes and Fitzgerald (1997); Novak, Murrow and Sinclair (1999).

4.2 Privacy

> *Three years ago Diane started her own consulting business. She has been so successful that she now has several people working for her and many clients. Their consulting work included advising on how to set up corporate intranets, designing database management systems, and advising about security.*
>
> *Presently she is designing a database management system for the personnel office of a medium-sized company. Diane has involved the client in the design process, informing the CEO, the director of computing, and the director of personnel about the progress of the system. It is now time to make decisions about the kind and degree of security to build into the system. Diane has described several options to the client. Because the system is going to cost more than they planned, the client has*

[2] Permission to copy all or part of this ACM paper is given at the end of that paper, provided proper ackowledgement is made.

decided to opt for a less secure system. She believes the information they will be storing is extremely sensitive. It will include performance evaluations, medical records for filing insurance claims, salaries, and so forth.

With weak security, employees working on client machines may be able to figure out ways to get access to this data, not to mention the possibility of on-line access from hackers. Diane feels strongly that the system should be much more secure. She has tried to explain the risks, but the CEO, director of computing and director of personnel all agree that less security will do. What should she do? Should she refuse to build the system as they request?

This case deals principally with the issue of privacy. The code deals with this in a number of ways. IT professionals are obliged to preserve the integrity and security of other's information (1.2) and to consider and respect people's privacy (4.2). Statement 2.6 also requires Diane advise her clients that the proposal is not in their best interests, which she rightly did. Her first obligation was to educate the company management as to their responsibilities to adequately protect the privacy of their employees; this may also be inferred from 5.3 about encouraging colleagues to pursue further professional development. If Diane failed to convince her client's management then she would need to consider her contractual obligations.

For further readings in this area, the following would be helpful: Hunter (1995); Kusserow (1995); Brankovic and Estivill-Castro (1999).

4.3 Confidentiality

Max works in a large state department of alcoholism and drug abuse. The agency administers programs for individuals with alcohol and drug problems, and maintains a huge database of information on the clients who use their services. Some of the data files contain the names and current addresses of clients.

Max has been asked to take a look at the track records of the treatment programs. He is to put together a report that contains the number of clients seen in each program each month for the past five years, length of each client's treatment, number of clients who return after completion of a program, criminal histories of clients, and so on. In order to put together this report, Max has been given access to all files in the agency's mainframe computer. After assembling the data into a file that includes the clients names, he downloads it to the computer in his office.

Under pressure to get the report finished by the deadline, Max decides he will have to work at home over the weekend in order to finish on time. He burns the information onto a CD and takes it home. After finishing the report he leaves the CD at home and forgets about it.

Whilst this case is similar to the one before it, in that it also deals with privacy, it deals more directly with the issue of confidentiality. The code used to have a principle (1.4) concerning the obligation to preserve the confidentiality of others' information, this has been omitted from the most recent publication of the code. Interestingly ACM code unlike ACS not only specifically addresses privacy, but also the closely related area of confidentiality; the ACM has specific clauses on constraining access to the type of data Max is dealing with, and on organisational leadership to ensure privacy obligations are adhered to within their organization. One can also presume that the

government agency Max works for has policies concerning protecting the identities of its clients. His relatives might accidentally discover the files and inappropriately use the information, which could damage the reputation of this agency's clients. One way around this problem for Max might have been to not have the names or other identifying information pertaining to clients in the report; a simple procedure such as this would have preserved the privacy and confidentiality of these clients to a significant extent (yet not all confidential data is of a kind that lends itself to this).

For people interested in further readings in this area, the following would be helpful: Weckert (1999); Brankovic and Estivill-Castro (1999) who discuss the difference as privacy being to do with people and confidentiality being about data, in the context of the Australian privacy act which protects federal government data ownership only [apparently there are no laws currently governing private corporation ownership of data].

4.4 Quality in Professional Work

> *A computer company is writing the first stage of a more efficient accounting system that will be used by the government. This system will save tax payers a considerable amount of money every year. A computer professional, who is asked to design the accounting system, assigns different parts of the system to her staff. One person is responsible for developing the reports; another is responsible for the internal processing; and a third for the user interface. The manager is shown the system and agrees that it can do everything in the requirements. The system is installed, but the staff finds the interface so difficult to use that their complaints are heard by upper-level management. Because of these complaints, upper-level management will not invest any more money in the development of the new accounting system and they go back to their original, more expensive system.*

The issues in this case are to do with quality and meeting user requirements. The code specifies that (2.1) products and services must meet operational requirements, that due regard needs to be shown to those affected by a system (4.4), and that IT professionals should strive to increase user feelings of personal satisfaction, competence and control (4.5). Presumably the failure to deliver a good quality product is the result of not following a quality assurance process. The problems with the interface would probably have come to light in a review process involving peers and/or users. The failure to implement a quality assurance process is a violation of ethical behaviour.

For further readings in this area of software quality, the following would be helpful: Collins and Miller (1995).

4.5 Fairness and Discrimination

> *In determining requirements for an information system to be used in an employment agency, the client explains that, when displaying applicants whose qualifications appear to match those required for a particular job, the names of white applicants are to be displayed ahead of those of non-white applicants, and the names of male applicants are to be displayed ahead of those of female applicants.*

The type of discrimination displayed in this case is addressed in a number of ways by the code. IT professionals are required to advise clients of potential conflicts on interest between assignments and legal or accepted community requirements (1.5). In fact, one is expected to make oneself

aware of such requirements (2.3). This clearly breaches the principle regarding "Social Implications", for instance respecting others and refraining from treating them unfairly (4.3). One might also see this as breaching professional conduct, as in refraining from knowingly engaging in, or being associated with, dishonest practices (6.2). In this case the designer of the system appears to be asked to build a system that favours white males and discriminates against non-whites and females. The code would suggest that the designer should point out the difficulties inherent in what the client is requesting and ask why this is being done. By making such a request the designer is being consistent with 1.5 (respecting the legalities involved), 6.6 and 6.9 (upholding and promoting the Code of Ethics).

The client might decide that they still want to go ahead and build a system that will enable them to favour white males. In this case the designer should refuse to build the system as proposed, as to do otherwise would be a clear violation of the code.

For people interested in further readings in this area, the following would be helpful: Hogan and James (1997).

4.6 Liability for Unreliability

> *A software development company has just produced a new software package that incorporates the new tax laws and figures taxes for both individuals and small businesses. The president of the company knows that the program has a number of bugs. He also believes the first firm to put this kind of software on the market is likely to capture the largest market share. The company widely advertises the program. When the company actually ships a CD, it includes a disclaimer of responsibility for errors resulting from the use of the program. The company expects it will receive a number of complaints, queries, and suggestions for modification. The company plans to use these to make changes and eventually issue updated, improved, and debugged versions. The president argues that this is general industry policy and that anyone who buys version 1.0 of a program knows this and will take proper precautions. Because of bugs, a number of users filed incorrect tax returns and were penalised by the ATO.*

One might view this scenario in the light of the imminent introduction of software by a number of vendors to help individuals and businesses deal with the GST.

In this case, given the company president was aware of the bugs, did not strive to achieve the highest quality product (2.1), and failed to inform customers about these bugs (3.1 and possibly 4.1), several principles of the code have been violated.

The risks to users are significant. They may be required to pay penalties for mistakes on their income tax declarations; mistakes resulting from this system. The company disclaimer was not made in good conscience and by doing this the company president is in effect violating the principle of accepting responsibility for one's work (2.5), which further encourages company staff to do likewise.

For further readings in this area, a recent discussion by Johnston and Acquaah-Gaisie (1999) looks at a variety liability issues from a legal standpoint.

4.7 Software Risks

> *A small software company is working on an integrated inventory control system for a very large national shoe manufacturer. The system will gather sales information daily*

from shoe stores nationwide. This information will be used by the accounting, shipping, and ordering departments to control all of the functions of this large corporation. The inventory functions are critical to the smooth operation of the system.

Jane, a quality assurance engineer with the software company, suspects that the inventory functions of the system are not sufficiently tested, although they have passed all their contracted tests. She is pressured by her employers to sign off on the software. Legally she is only required to perform those tests which have been agree to in the original contract. However, her considerable experience in software testing has led her to be concerned over risks of the system. Her employers say that they will go out of business if they do not deliver the software on time. Jane contends if the inventory subsystem fails, it will significantly harm their client and its employees. If the potential failure were to threaten lives, it would be clear to Jane that she should refuse to sign off. But since the degree of threatened harm is less, Jane is faced with a difficult moral decision.

Clearly Jane faces conflict in statement 2.1 between the financial needs of her employer and the operational needs of the client. This case again deals with product quality, as well as with professional integrity (2.6, 2.7). Jane rightly recognises that she should protect the client's needs (2.1, 2.4) and voice her concerns in an unbiased and objective manner (3.3). The code would suggest that Jane should not deliver a system she believes to be of a lower grade than necessary to meet the client's needs and that she should not mislead the client concerning the quality of the system. At very least, given the implications to her employer, the client ought to be informed about her reservations.

For further readings on software risks, the following would be helpful: Rahanu, Davies and Rogerson (1996).

4.8 Conflicts of Interest

A software consultant is negotiating a contract with a local community to design their traffic control system. He recommends they select the TCS system out of several available systems on the market. The consultant fails to mention that he is a major stockholder of the company producing TCS software.

A consultant in this situation ought to advise the client of the potential conflict of interest in connection with the work (1.5) and do so as soon as possible (1.6). The consultant may genuinely believe the TCS system is the best value for money (2.2), but principle 3 regarding "Honesty" would require disclosure up front. IT professionals need to ensure their clients are fully aware of their options and should refrain from professional recommendations modified for personal gain.

For further readings there is a philosophical look at this and related issues in Visala (1995).

4.9 Unauthorised Access

Joe is working on a project for his computer science course. The instructor has allotted a fixed amount of computer time for this project. Joe has run out of time, but has not yet finished the project. The instructor cannot be reached. Last year Joe worked as a student programmer for the campus computer center and is quite familiar with procedures to increase time allocations to accounts. Using what he learned last year, he is able to access the master account. Then he gives himself additional time and finishes his project.

Perhaps as a student Joe may not consider himself an IT professional, yet as a student member of the ACS he is expected to adhere to the code. Joe's actions violate several tenets of the code. The first tenet on "Priorities" says (1.2) members will preserve the security of and (1.3) will respect the proprietary nature of others' information (similarly 2.4). The tenet about computing professionalism also says (6.2) that members will not knowingly engage in dishonest practices.

For people interested in further readings in this area, the following would be helpful: Stallman (1995); Spafford (1995); Nitzberg (1998).

5. CONCLUSION

The aim of this paper has been to help IT professionals better understand how to apply the ACS Code of Ethics in their workplace. Ethical decision making often requires balancing many factors. In a small way case 7 illustrated conflict in ethical decision making for Jane, who had to choose between the financial needs of her employer and the operational needs of the client. Such conflicts require the IT professional to understand the spirit as well as the letter of the code. This is why this paper discussed the principles of the code before discussing specific cases studies. A more explicit approach to conflict resolution is that of the Joint IEEE-CS/ACM Task Force on Software Engineering Ethics and Professional Practice (SEERI, 1998) who recommended that in cases of conflict "public welfare" ought to be the overriding principle. Both the BCS and the ACM codes seem to place a greater emphasis on public welfare than does the ACS code. In a future revision of the ACS code it would be worth considering if members also believe that some ethical principles (such as public welfare) take precedence over others. If so, such precedence ought to clearly stated within the code. To foster further discussion in matters of ethical decision making, readers faced with special ethical computing situations are encouraged to share them with their colleagues, ACS staff, or with the Australian Institute of Computer Ethics (AICE, 1999).

The above discussion suggests some shortcomings in the code as it stands. One obvious one is to do with the omission of statement 1.4 on confidentiality. Perhaps this was done because confidentiality was seen as similar to privacy, but as shown above, they are separate. Confidentiality has to do with issues such as secure ("Secret", "Top Secret") documents and command in confidence materials. Confidentiality may also be about a society or community, such as the interests of the "Australian people". It ought to be explicitly referred to in the code.

Case 4, which dealt with "quality and professional work", showed up another difference in the ACS code to that of the ACM. The ACM code directly makes reference to good quality work, whereas the ACS code implies the need for good quality in one's professional work, without stating it explicitly. Perhaps this is just semantics, but in my view it would be worthwhile to publicly state (which is what the code does) that IT professionals value quality in their professional work.

As was discussed early in the paper, the ACS commissioned an Ethics Task Force to survey members attitudes to a number of ethics issues. That survey has been completed, with the analysis yet to be published. This is a positive step forward to promoting ethical thinking and to ensuring the ACS code remains relevant to IT professionals in the future.

Given the subjective nature of ethical decision making in particular situations, it is worth remembering the concluding statement of the ACS Standard of Conduct:

> *In summary, a member is expected to act at all times in a manner likely to be judged by informed, respected, and experienced peers in possession of all of the facts as the most ethical way to act in the circumstances.*

6. REFERENCES

ACS (2000): *IT Professional Recognition,* http://www.acs.org.au/news/aust-report.htm

ACS (1999a): *Code of Ethics,* http://www.acs.org.au/national/pospaper/acs131.htm

ACS (1999b): *About the Australian Computer Society,* http://www.acs.org.au/national/purpose.htm

ACS (1988a): *Code of Ethics Values and Ideals subscribed to by an ACS Member,* Melbourne: ACS Victorian Bulletin, September, 1-2.

ACS (1988b:) *Code of Ethics Values and Ideals subscribed to by an ACS Member,* Melbourne: ACS Victorian Bulletin, October, 18-19.

ACM (1999): *Code of Ethics and Professional Conduct,* Adopted by ACM Council 16/10/92, http://www.acm.org/constitution/code.html

AICE (1999): *Australian Institute of Computer Ethics,* http://www.aice.swin.edu.au/

ANDERSON, R.E., JOHNSON, D.G., GOTTERBARN, D. and PERROLLE, J. (1993): *Using the New ACM Code of Ethics in Decision Making,* Communications of the ACM, 36, (1): February, 98-107.

BARROSO ASENJO, P. (1997): Key Ethical Concepts for the Internet and for Ethical Codes of Computer Professionals, *The Australian Computer Journal,* 29(1):2-5.

BCS (1999): Code of Conduct, http://www.bcs.org.uk/aboutbcs/coc.htm

BRANKOVIC, L. and ESTIVILL-CASTRO, V. (1999): *Privacy Issues in Knowledge Discovery and Data Mining,* Australian Institute of Computer Ethics Conference, Lilydale: Swinburne University of Technology, July, 89-99.

CIFUENTES, C. and FITZGERALD, A. (1997): Copyright in Shareware Programs Distributed on the Internet, *The Australian Computer Journal,* 29(1): 24-30.

CLARKE, R. (1990): Social implications of IT – the professional's role, *The Australian Computer Journal,* 22(2): 27-29.

COLDWELL, R.A. (1987): Non-professional Practices in Computing: Some Thoughts on the Next Decade or So, *The Australian Computer Journal,* 19(4): 215-218.

COLLINS, W.R. and MILLER, K.W. (1995): *Programming and the Public Trust,* Computers, Ethics and Social Values, (Ed.) D.G. Johnson and H. Nissenbaum, New Jersey: Prentice Hall, 573-575.

EARLE, T.R. and FITZGERALD, E.P. (1983): Professionalism – Its Educational Aspects, *The Australian Computer Journal,* 15(3): 86-90.

HOGAN, J.M. and JAMES, P.C.J. (1997): Australian Approaches to Internet Content Regulation, *The Australian Computer Journal,* 29(1): 16-23.

HUNTER, L. (1995): *Public Image,* Computers, Ethics and Social Values, (Ed.) D.G. Johnson and H. Nissenbaum, New Jersey: Prentice Hall, 293-299.

JOHNSTON, S.W. and ACQUAAH-GAISIE 2, G. (1999): *Computers and Human Rights: Toward a UN code of computer ethics,* Australian Institute of Computer Ethics Conference, Lilydale: Swinburne University of Technology, July, 187-194.

KUSSEROW, R.P (1995): *The Government Needs Computer Matching To Root Out Waste And Fraud,* Computers, Ethics and Social Values, (Ed.) D.G. Johnson and H. Nissenbaum, New Jersey: Prentice Hall, 299-304.

LANGFORD, D. (1995): *Practical Computer Ethics,* London: McGraw-Hill Book Company.

LICHTENBERG, J. (1996): *What are Codes of Ethics for?,* Codes of Ethics and the Professions, (Ed.) M. Coady and S. Bloch, Melbourne: Melbourne University Press, 13-27.

MARTIN, M. and SCHINZINGER, R. (1995): *Codes of Ethics,* Computers, Ethics and Social Values, (Ed.) D.G. Johnson and H. Nissenbaum, New Jersey: Prentice Hall, 576-580.

MONTGOMERY, A.Y. (1987): The Australian Computer Journal: A Twentieth Anniversary Editorial Twenty Deadly Sins Against Professionalism and Progress in Australian Computing, *The Australian Computer Journal,* 19(4): 190-195.

MORRISON, P.R. and FORESTER, T. (1990): Teaching computer ethics and the social context of computing, *The Australian Computer Journal,* 22(2): 36-42.

NITZBERG, S. (1998): *Conflict and the Computer: Information Warfare and Related Ethical Issues,* Ethicomp98, Fourth International Conference on Ethical Issues of Information Technology, The Netherlands: Erasmus University, March, 498-507.

NOVAK, P., MURROW, J. and SINCLAIR, S. (1999): *Fair Use: Ethical Guidelines for the Educational Use of Copyrighted Material and the Internet,* Australian Institute of Computer Ethics Conference, Lilydale: Swinburne University of Technology, July, 265-275.

RAHANU, H., DAVIES, J. and ROGERSON, S. (1996): *Ethical Analysis of Software Failure Cases,* Volume 1, Proceedings of Ethicomp96, Third International Conference, Values and Social Responsibilities of the Computer Science, Spain: Pontificial University of Salamanca, November, 364-383.

SEERI (1998): Software Engineering Code of Ethics and Professional Practice Codes of Ethics, The Joint IEEE-CS/ACM Task Force on Software Engineering Ethics and Professional Practice recommended the adoption of this code of ethics to the ACM and the IEEE-CS, http://www-cs.etsu.edu/seeri/codes.htm.

SPAFFORD, E.H. (1995): *Are Computer Hacker Break-ins Ethical?,* Computers, Ethics and Social Values, (Ed.) D.G. Johnson and H. Nissenbaum, New Jersey: Prentice Hall, 125-135.

STALLMAN, R. (1995): *Are Computer Property Rights Absolute?,* Computers, Ethics and Social Values, (Ed.) D.G. Johnson and H. Nissenbaum, New Jersey: Prentice Hall, 115-119.

WECKERT, J. (1999): *Trust in the Information Age,* Australian Institute of Computer Ethics Conference, Lilydale: Swinburne University of Technology, July, 427-438.
WOODINGS, T. (1987): The Professional Development and Continuing Education of Computing Practitioners, *The Australian Computer Journal,* 19(4): 224-230.
VISALA, S. (1995): *Overcoming rivalling interests in systems development,* Volume 2, Proceedings of Ethicomp95, An International Conference on Ethical Issues of Using Information Technology, USA: De Montford University, March, 79-88.

7. APPENDIX

1. Priorities

I will serve the interests of my clients and employers, my employees and students, and the community generally, as matters of no less priority than the interests of myself or my colleagues.

1.1 I will endeavour to preserve continuity of computing services and information flow in my care.
1.2 I will endeavour to preserve the integrity and security of others' information.
1.3 I will respect the proprietary nature of others' information.
1.5[4] I will advise my client or employer of any potential conflicts of interest between my assignment and legal or other accepted community requirements.
1.6 I will advise my clients and employers as soon as possible of an conflicts of interest or conscientious objections which face me in connection with my work.

2. Competence

I will work competently and diligently for my clients and employers.

2.1 I will endeavour to provide products and services which match the operational and financial needs of my clients and employers.
2.2 I will give value for money in the services and products I supply.
2.3 I will make myself aware of relevant standards, and act accordingly.
2.4 I will respect and protect my clients' and employers' proprietary interests.
2.5 I will accept responsibility for my work.
2.6 I will advise my clients and employers when I believe a proposed project is not in their best interests.
2.7 I will go beyond my brief, if necessary, in order to act professionally.

3. Honesty

I will be honest in my representations of skills, knowledge, services and products.

3.1 I will not knowingly mislead a client or potential client as to the suitability of a product or service.
3.2 I will not misrepresent my skills or knowledge.
3.3 I will give opinions which are as far as possible unbiased and objective.
3.4 I will give realistic estimates for projects under my control.
3.5 I will not give professional opinions which I know are based on limited knowledge or experience.
3.6 I will give credit for work done by others where credit is due.

[3] Note, 1.4 has not been accidentally omitted. The current published version of the ACS code omits this principle, but keeps the numbering of the code as it was before. I could find no documented reason for the omission. The omitted principle is (ACS, 1988b): *1.4 I will endeavour to preserve the confidentiality of others' information.* As can be seen in the case studies within this paper, there is no equivalent to this omitted statement in the remainder of the code.

4. Social Implications

I will strive to enhance the quality of life of those affected by my work.

4.1 I will protect and promote the health and safety of those affected by my work.

4.2 I will consider and respect people's privacy which might be affected by my work.

4.3 I will respect my employees and refrain from treating them unfairly .

4.4 I will endeavour to understand. and give due regard to, the perceptions of those affected by my work, whether or not I agree with those perceptions.

4.5 I will attempt to increase the feelings of personal satisfactions, competence, and control of those affected by my work.

4.6 I will not require, or attempt to influence, any person to take any action which would involve a breach of this Code.

5. Professional Development

I will enhance my own professional development, and that of my colleagues, employees and students.

5.1 I will continue to upgrade my knowledge and skills.

5.2 I will increase my awareness of issues affecting the computing profession and its relationship with the community.

5.3 I will encourage my colleagues, employees and students to continue their own professional development.

6. Computing Profession

I will enhance the integrity of the Computing Profession and the respect of its members for each other.

6.1 I will respect, and seek when necessary, the professional opinions of colleagues in their areas of competence.

6.2 I will not knowingly engage in, or be associated with, dishonest or fraudulent practices.

6.3 I will not attempt to enhance my own reputation at the expense of another's reputation.

6.4 I will co-operate in advancing information processing by communication with other professionals, students and the public, and by contributing to the efforts of professional and scientific societies and schools.

6.5 I will distance myself professionally from someone whose membership of the ACS has been terminated because of unethical behaviour or unsatisfactory conduct.

6.6 I will take appropriate action if I discover a member, or a person, who could potentially be a member of the ACS, engaging in unethical behaviour.

6.7 I will seek advice from the ACS when faced with an ethical dilemma I am unable to resolve by myself.

6.8 I will do what I can to ensure that the corporate actions of the ACS are in accordance with this Code.

6.9 I acknowledge my debt to the Computing Profession and in return will protect and promote professionalism in computing.

BIOGRAPHICAL NOTES

Oliver K. Burmeister is a lecturer in the School of Information Technology at Swinburne University of Technology. His research interests focus on Human Factors in IT. He was a founding member of the Australian Institute of Computer Ethics and is currently completing a Master of Information Technology specialising in Human-Computer Interaction.

Partitioning Number Sequences into Optimal Subsequences

J. Zobel

Department of Computer Science
RMIT University
GPO Box 2476V
Melbourne 3001 Australia
email: jz@cs.rmit.edu.au

P. Dart

Department of Computer Science and Software Engineering
The University of Melbourne
Parkville 3052 Australia
email: philip@cs.mu.oz.au

We consider how to partition finite sequences of positive numbers into subsequences such that each resulting subsequence has a sum of at least a given minimum. Given several different optimality criteria, based on maximising the number of subsequences and minimising their variance in size, we develop and analyse a series of algorithms that yield optimal solutions.

Keywords: algorithms, analysis of algorithms, computational complexity.

1. INTRODUCTION

Consider a finite sequence of positive numbers that is to be partitioned into subsequences as follows. Each subsequence must contain numbers whose sum is at least L, where L is fixed; the variation between the sums is to be kept small; and the sums should be close to L. That is, the criteria for the partitioning are that the number of sums should be maximised and the variance should be minimised.

The problem of partitioning arose from our work in information retrieval, where it can be desirable to divide documents consisting of small units of text (such as sentences or paragraphs) into larger units of consistent length, for retrieval or presentation (Kaszkiel *et al,* 1999; Zobel *et al,* 1995). First, enforcing a minimum length helps ensure that there is sufficient content to allow accurate matching of queries and documents. Second, when units of text significantly vary in length, estimation of their relevance to a query is inaccurate, so it is desirable to reduce the variance. Partitioning can also be used for division of irregularly-sized data such as web pages into packets for network transmission, where there is a trade-off between response time and total cost of transmission.

Similar problems have been described by other authors. Larmore and Hirschberg (1985) have considered how to break scrolls into pages whose lengths must fall within a certain range. In

Copyright© 2000, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the JRPIT copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

Manuscript received: June 2000
Editor: John Roddick

contrast to partitioning, any successful pagination is satisfactory. Knuth and Plass (1981) have comprehensively discussed the problems presented by breaking paragraphs into lines with acceptable inter-word spacing, a far more complex problem than partitioning, for not only can the size of the items in a line be varied within certain limits, but the items can be divided with a hyphen between two lines. Another related problem is bin packing, in which items are to be placed in the minimum number of bins of some fixed maximum capacity (Garey and Johnson, 1979); however, packing does not have a constraint on the order of items.

2. PARTITIONING

Partitioning can be described formally as follows. Given a *source* sequence, that is, a sequence of positive numbers

$$W = \langle w_1, \ldots, w_n \rangle ,$$

we wish to coalesce adjacent numbers in the sequence to form a *target* sequence, that is, a new sequence of positive numbers

$$T = \langle t_1, \ldots, t_m \rangle .$$

A target is such that each $t_i \geq L$, where L is a fixed lower bound satisfying $\sum_{j=1}^{n} w_j \geq L > 0$, and the target's elements are defined by

$$t_i = \sum_{j=k_{i-1}+1}^{k_i} w_j ,$$

where $k_{i-1} < k_i$, $k_0 = 0$ and $k_m = n$. Thus the ith subsequence consists of numbers $w_{k_{i-1}+1}, \ldots, w_{k_i}$ and has sum t_i.

In addition, we wish to choose $k_1, \ldots, k_{m-1}$ to minimise the average distance between each t_i and L. A suitable function for minimising the average distance between t_i and L is the variance from L given by

$$variance(T) = \frac{1}{m} \sum_{i=1}^{m} (t_i - L)^2 .$$

Before we describe our algorithms for finding the optimal target, we consider some of the difficulties presented by partitioning. One is that local optimality does not imply global optimality. For example, consider a source

$$\langle 10,1,9,2,8,3,7,4 \rangle$$

where L is 10. Two targets are $\langle 10,10,10,14 \rangle$ and $\langle 11,11,11,11 \rangle$. The first target is locally better at the left-hand end, but the second is optimal. Thus an algorithm for finding an optimal target cannot proceed by accruing numbers in the source and choosing the optimal partial target.

Another difficulty presented by partitioning is that the target with lowest variance may not be the longest possible. An example source with this property is

$$\langle 10,5,5,9,9,5,5,9,9,5,5,9,9,5,5,9,9,5,5,9,9,5,5,10 \rangle .$$

where L is 10. Two targets are

$$\langle 10,10,18,10,18,10,18,10,18,10,18,10,10 \rangle .$$

and

$$\langle 15,14,14,14,14,14,14,14,14,14,14,14,15 \rangle .$$

The latter is shorter, but has lower variance. However, it is difficult to construct such sources, and we believe them to be very rare. Note that if the source is shortened by omission of a ⟨5,5,9,9⟩ subsequence, the longer target has the lower variance; that is, the source given is the shortest (of that pattern) that results in lower variance for the shorter target.

3. OPTIMISING FOR VARIANCE

The optimum target, with regard to any optimality criteria, can be found exhaustively as follows. Any given target can be written as

$$T = \langle w_1 + \ldots + w_{k_1}, w_{k_1+1} + \ldots + w_{k_2}, \ldots, w_{k_{m-1}+1} + \ldots + w_{k_m} \rangle .$$

From this form it can be seen that all possible targets can be found by enumerating combinations of sums, and that there are 2^{n-1} different combinations; note, however, that some of these combinations can contain some $t_i < L$ and are therefore not targets.

1. Set $S_1 \leftarrow \{\langle w_1 \rangle\}$.
2. For each j from 2 to n,
   (a) Create an empty set S_j.
   (b) For each $l \in S_{j-1}$,
      i. Add $l \cdot w_j$ to S_j.
      ii. Add $l + w_j$ to S_j.
3. Set $S \leftarrow \{l \in S_n | v \geq L$ for each value v in $l\}$.
4. Choose from S the sequence $l \in S$ with minimum variance.

Figure 1: Algorith A – naive optimisation for variance

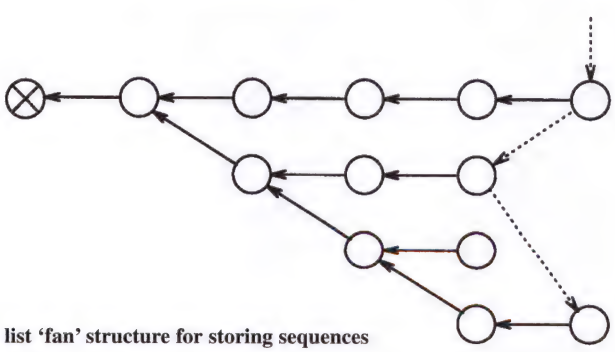


Figure 2: Linked list 'fan' structure for storing sequences

We use the following notation. Suppose $l = \langle v_1,\ldots,v_k \rangle$ is a sequence and v is a value. Then a *composition* $\langle v_1,\ldots,v_k+v \rangle$ is denoted by $l+v$; an *extension* $\langle v_1,\ldots,v_k, v \rangle$ is denoted by $l \cdot v$; the length k of l is denoted by *len(l)*; and the last value v_k in l is denoted by *last(l)*.

The set S of targets can be enumerated by the method shown in Figure 1. Input are n and a source $\langle w_1,\ldots,w_n\rangle$. We assume, as in all the algorithms in this paper, that $\sum_{j=1}^{n} w_j \geq L$. That is, we assume there is a valid target.

A simple way to implement Algorithm A (and Algorithms B and C, described later) is to have a linked structure as follows. Each t_k value requires a node with fields containing: its value; k; the sum $\sum_{i=1}^{k}(t_i - L)^2$; a pointer to t_k-1; and a pointer to the last t value in another sequence. The inter-sequence pointers give a linked list joining the ends of all the candidate sequences together. Candidate sequences can be reconstructed by following intra-sequence pointers from the end of the sequence to the start. To simplify implementation, a sentinel node should be placed before w_1. This 'fan' structure is illustrated in Figure 2.

In Figure 2, the dashed lines are the inter-sequence pointers and the crossed node is the sentinel. One sequence is unreachable, because there is no pointer to the last node; although not possible with Algorithm A, the other algorithms can create such a structure.

Using the fan structure, finding the optimal sequence – the sequence with the lowest variance – is straightforward: the information stored in the last node in each sequence can be used to derive the variance, and the last nodes are connected by the inter-sequence pointers. The fan structure itself is desirable because any leading subsequence that is common to several sequences is stored once only.

Assuming the fan implementation, we can derive the following properties.

Theorem 1 Algorithm A has space and time complexity of $O(2^n)$.

Proof At step 2b of Algorithm A, two sequences are added to S_j for each sequence in S_{j-1}, so that $|S_j|=2|S_{j-1}|$. Given the fan implementation, the additional cost in space at each j is linear in the current number of sequences. The result follows. □

Theorem 2 For any source W and the set S yielded by application of Algorithm A to W, the target in S with the lowest variance will be optimal.

Proof The result follows from the observation that all targets are in S. □

We now show how Algorithm A can be modified to yield the more efficient Algorithm B, described below. The inefficiencies in Algorithm A can be addressed using the following propositions.

Proposition 1 Consider S_j, for any j where $1 \leq j \leq n$, derived by Algorithm A. For any sequence $l = \langle t_1,\ldots,t_{k-1},t_k\rangle \in S_j$ such that $t_{k-1} < L$, no sequence in S_n derived from l, by composition or extension, is a target.

Proof Straightforward. □

Thus subsequences that cannot lead to targets can be progressively eliminated, by modifying step 2(b)i of Algorithm A such that $l \cdot w_j$ is added to S_j only if $last(l) \leq L$. It can be seen by induction that, for any j and any sequence $\langle t_1,\ldots,t_k\rangle \in S_j$, each $t_i \geq L$ for $1 \leq i < k$. Step 3 can then be modified to keep only the sequences whose last element is at least L.

Proposition 2 Consider two targets derived by Algorithm A,

$$T = \langle t_1,\ldots,t_k + w_j + v, t'_1,\ldots,t'_{k'}\rangle \in S \text{ and}$$
$$T' = \langle t_1,\ldots,t_k, w_j + v, t'_1,\ldots,t'_{k'}\rangle \in S.$$

Then the variance of T' is less than the variance of T.

Proof Straightforward. □

1. Set $S_1 \leftarrow \{\langle w_1 \rangle\}$.
2. For each j from 2 to n,
   (a) Create an empty set S_j.
   (b) For each $l \in S_{j-1}$,
      i. If $last(l) < L$ or $w_j < L$, add $l + w_j$ to S_j.
      ii. If $last(l) \geq L$, add $l \cdot w_j$ to S_j.
3. Set $S \leftarrow \{l \in S_n | last(l) \geq L\}$.
4. Choose from S the sequence $l \in S$ with minimum variance.

Figure 3: Algorithm B – optimisation for variance.

It follows that T – or, if $w_j \geq L$, even its leading subsequence $l = \langle t_1,\ldots,t_k+w_j \rangle$ – need never be created, since for any target derived from l there will be a target with lower variance derived from $\langle t_1,\ldots,t_k+w_j \rangle$. Sequences such as l can be avoided by not applying composition when $last(l) \geq L$ and $w_j \geq L$.

The algorithm can now be restated as Algorithm B, shown in Figure 3, and again the fan implementation should be used. As for Algorithm A, the sequence in S with the lowest variance will be the optimal target.

The complexity is as follows.

Theorem 3 Algorithm B has space and time complexity of $O(2^n)$.

Proof The result follows from the observation that in the worst case Algorithm B generates the same sequences as Algorithm A, and never generates more sequences. □

Although the upper bound on the cost of Algorithm B is the same as that of Algorithm A, Algorithm B is in general cheaper, particularly when the source contains elements that are at least L. Nonetheless, for some sources Algorithm B has cost $O(2^n)$.

4. OPTIMISING FOR LENGTH AND VARIANCE

We can use length and variance constraints to refine Algorithm B in accordance with the following proposition, to yield an Algorithm C that produces targets of maximum length, and of these targets, chooses the target with minimum variance.

Proposition 3 Consider two sequences

$$l = \langle t_1, \ldots, t_k \rangle \in S_{j-1} \text{ and}$$
$$l' = \langle t'_1, \ldots, t'_{k'} \rangle \in S_{j-1},$$

where $t_k \geq L$ and $t'_{k'} \geq L$.

1. For any target
$$T = \langle t_1, \ldots, t_k, w_j + v, u_1, \ldots, u_p \rangle$$
that contains l as a leading subsequence, there is a target
$$T' = \langle t'_1, \ldots, t'_{k'}, w_j + v, u_1, \ldots, u_p \rangle$$
that contains l' as a leading subsequence.
2. If $k < k'$ then T will be shorter than T'.
3. If $k = k'$ and $variance(l) > variance(l')$, then $variance(T) > variance(T')$.
4. If $k = k'$ and $variance(l) = variance(l')$, then $variance(T) = variance(T')$.

Proof Straightforward. □

Thus, when choosing sequences in S_{j-1} for extension, only the sequence with maximal length and minimal variance need be considered at step 2(b)ii.

Similarly, step 2(b)i of Algorithm B can be refined as follows. If $l \in S_{j-1}$, $last(l) \geq L$, and $w_j \geq L$, then, as discussed above, for each target T that can be derived from $l + w_j$, a target T' with lower variance can be derived from $l \cdot w_j$. Moreover, T will be shorter than T'. Furthermore, suppose that S_{j-1} contains two sequences $l = \langle t_1, \ldots, t_k \rangle$ and $l' = \langle t'_1, \ldots, t'_{k'} \rangle$, where $t_k \geq L$, $t'_{k'} \geq L$, and $k < k'$. From the above proposition, for each target that can be derived from l, a longer target can be derived from l', so we do not need to consider composition to l.

Using the refinements, an algorithm that optimises for length and variance can be stated as in Figure 4.

Theorem 4 Consider the set S returned by Algorithm C.
1. Each sequence in S is a target of maximal length.
2. The target of maximal length and, for that length, minimal variance, is in S.

Proof The result follows from Proposition 3 and from case analysis of the algorithm. □

To analyse the complexity of Algorithm C we first need two lemmas.

Lemma 1 In Algorithm C, $|S_j| \leq |S_{j-1}| + 1$.

Proof In step 2e of Algorithm C, at most one sequence is added to S_j for each sequence in S_{j-1}, and only one sequence is added to S_j at step 2d. No other steps add sequences. □

Lemma 2 In Algorithm C, if two adjacent source elements w_{j-1} and w_j are both at least L, the set S_j has just one element.

Proof Neither option of step 2e applies, and step 2d adds exactly one candidate. □

We say that a sequence is *minor* if it does not contain two adjacent elements of at least L.

Theorem 5 Algorithm C has time complexity of $O(ns)$, where s is the length of the longest minor subsequence in the input source.

Proof The result follows from Lemmas 1 and 2. □

1. Set $S_1 \leftarrow \{\langle w_1 \rangle\}$.
2. For each j from 2 to n,
   (a) Create an empty set S_j.
   (b) Set $M \leftarrow \{l \in S_{j-1} | last(l) \geq L\}$.
   (c) Let K be the length of the longest sequence in M.
   (d) If $M' = \{l \in M | len(l) = K\}$ is not empty, choose an arbitrary sequence $l \in M'$ such that there is no sequence $l' \in M'$ with lower variance, and add $l \cdot w_j$ to S_j.
   (e) For each $l \in S_{j-1}$,
      i. If $last(l) < L$ add $l + w_j$ to S_j.
      ii. If $last(l) \geq L$ and $len(l) = K$ and $w_j < L$ then add $l + w_j$ to S_j.
3. Set $S \leftarrow \{l \in S_n | last(l) \geq L\}$.
4. Choose from S the sequence $l \in S$ with minimum variance.

Figure 4: Algorithm C – optimisation for length and variance.

Algorithm C should be implemented with the following modifications to the fan structure. At step 2d, a new node for w_j must be created, and must point to the node for the last element in l; the node for this element must be marked as referenced. At step 2e, each value *last(l)* can be replaced by *last(l)* + w_j, except for the single value of *last(l)* that was marked as referenced, in which case a node for *last(l)* + w_j must be created, with a pointer to *last(l)*'s predecessor.

Theorem 6 Algorithm C has space complexity of $O(n)$.

Proof Using the fan structure, at most two nodes are added for each element in the source. □

5. OPTIMISING FOR LENGTH ALONE

Algorithm D, shown in Figure 5, is linear in time and space and emits the solution as it proceeds. The variables *prev* and *curr* refer to the previous and current target sequence entries respectively.

1. Set *prev* $\leftarrow$ *undefined* and *curr* $\leftarrow w_1$.
2. For each j from 2 to n,
   (a) If *curr* $\geq L$, emit *prev* if it is defined, set *prev* $\leftarrow$ *curr*, and set *curr* $\leftarrow w_j$.
   (b) Otherwise, if $w_j >$ *prev* then set *prev* $\leftarrow$ *prev* + *curr* and set *curr* $\leftarrow w_j$.
   (c) Otherwise, set *curr* $\leftarrow$ *curr* + w_j.
3. If *curr* $< L$ then set *prev* $\leftarrow$ *prev* + *curr* and emit *prev*. Otherwise, emit *prev* followed by *curr*.

Figure 5: Algorithm D – optimisation for length.

Theorem 7 Algorithm D has $O(n)$ time complexity, requires $O(n)$ space to store the solution but only $O(1)$ working space.

Proof The algorithm requires at most four comparisons and one sum for each element of the source, and only *curr* and *prev* are stored.

Moreover, Algorithm D emits partial solutions as it proceeds, whereas the other algorithms do not emit any solution until the whole source has been inspected.

Theorem 8 Algorithm D always generated targets of the maximum length.

Proof Step 2 generates leading subsequences in which, including the values of *prev* and *curr*, all but the last element is at least L. Using induction on j, it can be shown that, for a source of any length n, the partial target generated by step 2 is of the longest possible length, and that if $curr < L$ then *curr* is at least as large as the last element in any partial target of the same length. That the target generated by Algorithm D is of maximal length follows immediately.

However, the targets computed by Algorithm D do not necessarily have minimal variance for their length. We believe that there is no linear time algorithm that yields such targets, because optimisation of variance cannot be determined locally.

6. SUMMARY

We have described four algorithms for computing an optimal partitioning of a source into a target, for three different optimality criteria. The first two algorithms, A and B, minimise variance, at a complexity of $O(2^n)$ for a source of length n; we believe it unlikely that there is an algorithm that minimises variance at lower complexity. The third algorithm, C, minimises variance at maximum target length and has space complexity of $O(n)$ and time complexity of $O(ns)$, where s is the length of the longest minor subsequence in the source. Thus the worst case is a source for which all elements are less than L, giving a complexity of $O(n^2)$ – a marked improvement over the first two algorithms. Given our observation that almost all targets of minimum variance are of maximum length, it follows that Algorithm C is an excellent heuristic method for minimising variance alone. The final algorithm, D, maximises target length and has space and time complexity of $O(n)$. In the application of interest to us – dividing long documents into pages – there were sources of over ten thousand elements. In experiments with a large collection of documents represented as sequences of elements, Algorithms A and B were impossible to use because of the space requirements, while Algorithm C was acceptable but occasionally slow. Algorithm D took less than a second and yielded no solutions with unacceptable variance, but in some documents the variance was much greater than in the solutions produced by the other algorithms.

ACKNOWLEDGEMENTS

We wish to thank Alistair Moffat, James Thom, and Nick Wormald. This work was supported by the Australian Research Council and the Collaborative Information Technology Research Institute in Melbourne, Australia.

REFERENCES

GAREY, M.R. and JOHNSON D.S. (1979): *Computers and Intractability.* San Francisco, Freeman.

KASZKIEL, M., ZOBEL, J. and SACKS-DAVIS, R. (1999): Efficient passage ranking for document databases. *ACM Transactions on Information Systems,* 17(4) October: 406–439.

KNUTH, D.E. and PLASS, M.F. (1981): Breaking paragraphs into lines. *Software – Practice and Experience,* 11:1119–1184.

LARMORE, L.L. and HIRSCHBERG, D.S. (1985): Efficient optimal pagination of scrolls. Communications of the ACM, 28(8): 854–856.

ZOBEL, J., MOFFAT, A., WILKINSON, R. and SACKS-DAVIS, R. (1995): Efficient retrieval of partial documents. Information Processing & Management, 31(3): 361–377.

BIOGRAPHICAL NOTES

Associate Professor Justin Zobel received his PhD from the University of Melbourne, where he worked for several years as a lecturer in the Department of Computer Science. Since 1990 he has been in the Department of Computer Science at RMIT. In the research community, he is best known for his role in the development of algorithms for efficient text retrieval. He is also an active researcher in the areas of database systems, bioinformatic retrieval systems, compression and research methods.

Philip Dart is a senior lecturer in the Department of Computer Science and Software Engineering at the University of Melbourne. He received his PhD from the University of Melbourne and returned, in 1992, as Software Engineering coordinator after several years as a research scientist in the Defence Science and Technology Organisation. While his main research focus has been in software engineering, he has worked in other areas including text retrieval.

AICE2000 COMPUTER ETHICS CONFERENCE

The Australian Institute of Computer Ethics (AICE) is holding it second conference, AICE2000, in Canberra on 11–12 November this year, immediately following the 2000 Information Outlook (IO2000) Conference of the Canberra Branch of the Australian Computer Society (ACS), and at the same venue (ANU).

AICE2000 is being hosted by the Centre for Applied Philosophy and Public Ethics, Charles Sturt University, an ARC funded Special Research Centre, the Australian Catholic University and the University of Canberra.

The theme of the conference is 'Computer Ethics, why bother?' and the keynote speaker is Professor Jeroen van den Hoven of Erasmus University, Rotterdam, one of the leading computer ethicists in Europe.

Participants who attend both AICE2000 and IO2000 will receive a discount.

AICE2000 is endorsed by the ACS.

Visit the AICE2000 website: http://www.ise.canberra.edu.au/AICE2000

For further information contact John Barlow at j.barlow@mary.acu.edu.au or phone (02) 9739 2839.

POSTGRADUATE COURSES IN INFORMATION SECURITY

Midyear intake available to coursework based Graduate Certificate, Graduate Diploma and Master's Information Security courses at RMIT.

See our website at:
http://www.ma.rmit.edu.au/kepler/pgrad/InfoSec.html

or call (03) 9925 2283

or email serdar.rmit.edu.au

A Slicing-Based Approach to Enhance Petri Net Reachability Analysis*

W.J. Lee and H.N. Kim

Software Engineering Department, ETRI
161 Kajong Dong, Yusong Gu, Taejon 305–350, South Korea
Email: {woojin,hnkim}@etri.re.kr

S.D. Cha

Department of EECS and Advanced Information Technology Research Center (AITrc)
Korea Advanced Institute of Science and Technology
373–1, Kusong Dong, Yusong Gu, Taejon 305–701, South Korea
Email: cha@salmosa.kaist.ac.kr

Y.R. Kwon

Department of Electrical Engineering & Computer Science (EECS)
Korea Advanced Institute of Science and Technology
373–1, Kusong Dong, Yusong Gu, Taejon 305–701, South Korea
Email: kwon@salmosa.kaist.ac.kr

Reachability graph analysis is one of the most widely used techniques to verify the behaviour of asynchronous and concurrent systems modeled in Petri nets. Unfortunately, a state explosion problem is often the bottleneck when applying Petri net modeling and analysis to large and complex industrial systems. This paper proposes an approach in which Petri net slices are computed based on structural concurrency inherent in the P/T net and compositional reachability graph analysis is performed. Petri net slices are proven to provide behavioural equivalence to P/T nets. This approach may enable verification of properties such as boundedness and liveness which may fail on unsliced P/T nets due to a state explosion problem. Effectiveness and scalability of our approach is demonstrated using both dining philosophers and feature interaction problems found in telecommunication software.

Key words and Phrases: Petri nets, Place/Transition nets, reachability analysis, Petri net Slice, compositional analysis, structural concurrency.

Classification of the paper: Software-Software Engineering-Software/Program Verification-Formal Methods

1. INTRODUCTION

Petri nets are widely used to model and verify the behaviour of asynchronous systems (Suzuki *et al*, 1990), concurrent systems, and real-time systems (Ghezzi *et al*, 1991; Bucci and Vicario, 1995). Petri net-based formalisms have been significantly extended to enhance expressiveness and to

* This work was partially supported by Korea Science and Engineering Foundation (KOSEF) through the Advanced Information Technology Research Centre (AITrc).

Copyright© 2000, Australian Computer Society Inc. General permission to republish, but not for profit, all or part of this material is granted, provided that the JRPIT copyright notice is given and that reference is made to the publication, to its date of issue, and to the fact that reprinting privileges were granted by permission of the Australian Computer Society Inc.

Manuscript received: March 2000
Associate Editor: Hugh Williams

develop powerful and automated analysis techniques. Place/Transition (P/T) nets are often regarded as low-level Petri nets, and extensions include high-level Petri nets such as Jensen's (1992) colored Petri nets(CPN), object-oriented Petri nets (Battiston and Cindio, 1993; Perkusich and Figueiredo, 1997), Lee *et al* (1998) constraints-based modular Petri nets(CMPNs) etc. Although motivations vary significantly, the majority of extended Petri net-based formalisms are compatible in that they can be converted into semantically equivalent P/T nets and that verification techniques developed for P/T nets such as reachability analysis, invariants analysis (Murata, 1989), net unfoldings (Esparza, 1993) can be applied on extended Petri net formalisms.

Reachability analysis is the most frequently employed technique to examine the intended system behaviour. Although simple and straightforward in concept, reachability analysis is often considered impractical due to a state explosion problem when applied to nontrivial and real world problems. Several approaches have been proposed to try minimizing the number of system states when generating a reachability graph. Reduction rules (Murata, 1989) were proposed to reduce the size of the original model. Jorgensen (1997) demonstrated the potential of verification based on state spaces reduced by equivalence relation. And compositional analysis approaches reduce the possibility of state explosion by incrementally generating reachable states, which was applied to Modular Petri nets (Damm *et al,* 1989; Valmari, 1990; Notomi and Murata, 1994; Christensen and Petrucci, 1995; Schreiber, 1995). Unfortunately, compositional analysis, while an attractive option, may not be applied directly on P/T nets since a system model must be composed as a set of modular Petri nets. However, the majority of industrial applications of Petri nets still rely on classical P/T nets or high-level extensions such as Colored Petri nets, and there are no systematic approaches to convert a P/T net or CPN into modular Petri nets.

Our work is an attempt to enable modelling and analysis of P/T nets on large and complex industrial systems which previously could not be analysed due to the state explosion problem. By developing partitioning criteria and a slicing algorithm for dividing a huge P/T net model into meaningful and manageable modules, the compositional analysis technique may be applicable to P/T net models. In order to enhance the efficiency of compositional reachability analysis, modules should be defined by considering cohesive internal dependency and independency among modules. A slicing algorithm has a role to find the concurrent units and to partition a huge model into Petri net slices while preserving the behaviours of the original model by using concurrent units. Reachability analysis of Petri net slices can be performed in a compositional approach since Petri net slices are modules with synchronising by shared transitions. An S-invariant property can be used in analysing boundedness checking since each Petri net slice is deduced from S-invariant.

This paper is structured as follows. Section 2 presents the concurrency in P/T nets and defines structural concurrency and concurrent units. In section 3, we propose a slicing algorithm which finds concurrent units and partitions the original model into Petri nets slices. And we prove that the original model and Petri nets slices have the same behaviours. We introduce the compositional reachability analysis technique by using Petri net slices in section 4. In order to show effectiveness and practicability of our approach, in section 5, we describe the dining philosophy problem and feature interaction problems of telecommunication systems by Petri net slices. Finally, in section 6, we provide conclusions and future work.

2. CONCURRENT UNITS IN P/T NETS

Petri nets have been used for describing the concurrent properties of a system. In a Petri net model, the concurrency is represented by relationships among transitions which are simultaneously enabled without any causal dependency. Figure 1 shows a P/T net model for the dining philosopher problem.

After spending time thinking, a philosopher takes a left and right fork in arbitrary order, and eats spaghetti. To avoid deadlock situations in the dining philosopher problem, we add an additional functionality into the model: although a philosopher has taken a fork, he or she can abandon trying to take the other fork, release the taken fork, and return to the thinking state.

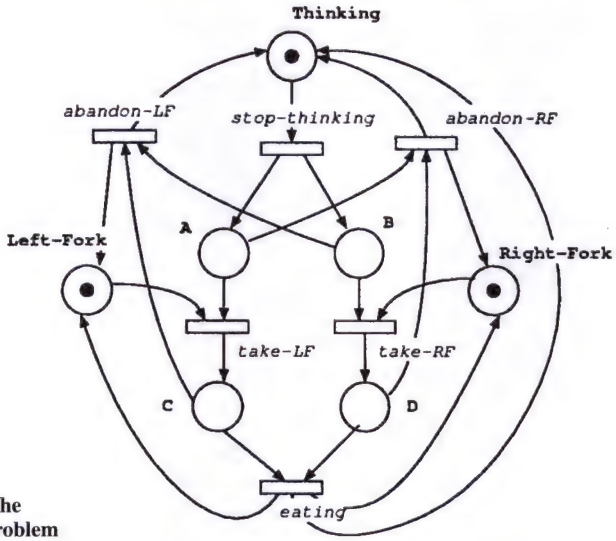


Figure 1:
A P/T net model for the dining philosopher problem

Concurrency in Petri net models are divided into structural concurrency and token concurrency. Structural concurrency is a behavioural characteristic appearing in structural connections among places and transitions. In Figure 1, transitions *take-LF* and *take-RF* are simultaneously firable if places A, B, Left-Fork, and Right-Fork have tokens. Finding structural concurrency is performed by checking the causality in the model without considering the markings of places. If two transitions have no causal relationship, the two transitions are structurally concurrent. Token concurrency is not relevant to the structural architecture but markings of places. If places Thinking, B, and Right-Fork have tokens, transitions *stop-thinking* and *take-RF* are concurrently firable, although they have causal dependency. Usually, a Petri net model may have both structural concurrency and token concurrency simultaneously. However, we consider only structural concurrency in partitioning the model since token concurrency may appear among all the pairs of transitions on the assumption that all the places in the model have sufficient tokens.

In a Petri net model, there may be tokens in several places, which have their own control threads. Since transitions may join several control threads by collecting incoming arcs or may divide a single control thread by distributing several outgoing arcs, these control threads are complicatedly interrelated. Although a token interacts with other ones by transitions, its trajectory can be deterministically and independently definable.

A concurrent unit can be detected by a trajectory of a virtual token, which means that a trajectory may start from a virtual token of an arbitrary place, not always from an initial token. Trajectories of virtual tokens may be obtained by using S-invariant concepts (Reisig, 1985), which represent the sets of places which do not change their token count during transition firings. A trajectory of a token is a special S-invariant where the token count is one. Figure 2 shows an example of concurrent units in the dining philosopher model. The first concurrent unit is the trajectory of a token in Left-Fork,

which is the trajectory of the left fork, and the second one is the trajectory of a token in Thinking, which means that a philosopher handles the right fork.

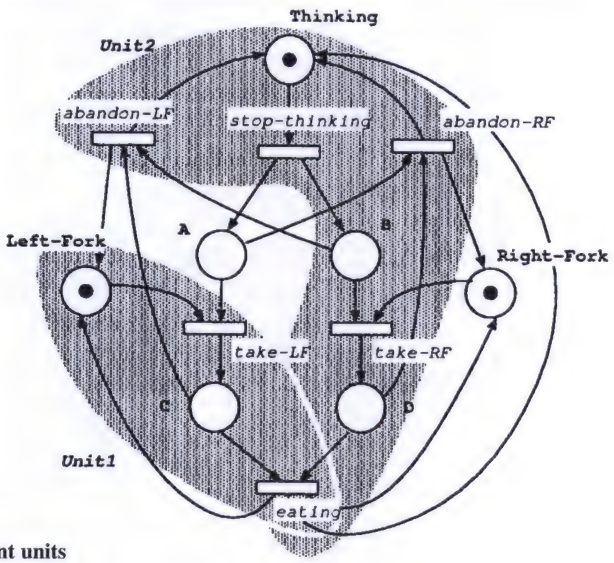


Figure 2:
An example of concurrent units

S-invariants are formally defined as follows. In general, the arc weights are positive integer and default arc weight is 1.

Definition 2.1 S-invariants
Let N be a P/T net. A place vector, $i : S_N \rightarrow \mathbf{Z}$ is called an S-invariant of N iff $\underline{N}' \bullet i = 0$, where $\underline{N}'$ is a matrix representation of N (transitions x *places).*

Let the number of places in a Petri net model be n. The solution of the matrix equation, $\underline{N}' \bullet i = 0$, can be obtained in time complexity $O(n^3)$ (Reisig, 1985). Solution vectors, that is, S-invariants may include negative integers. And two solution vectors may be merged to be another one. Special S-invariants which can be used in constructing concurrent units, *minimal invariants,* are defined as follows.

Definition 2.2 *Minimal Invariants*
Positive S-invariants which are not contained in other S-invariants are called minimal invariants.

In Figure 2, $Unit_1$, $Unit_2$, {*Right–Fork, D*} and {*Thinking, A, C*} are all minimal invariants.

3. SLICING A PETRI NET MODEL

A Petri net model is partitioned into concurrent units using minimal invariants. In order to preserve all the information in the original model, uncovered places should be added into minimally-connectable concurrent units since minimal invariants may not cover all the places.

```
SliceSet = φ;
SetofInvariant = find_minimal_invariants(N);
do {
    small_invariant = find_smallest_invariant(SetofInvariant);
    SliceSet = SliceSet ∪ { small_invariant };
    SetofInvariant = SetofInvariant - { small_invariant };
} until ( Place(SetofInvariant) ⊆ Place(SliceSet) OR Place(N) == Place(SliceSet) );
if ( Place(SliceSet) ≠ Place(N) ) {
    Uncovered_Place_Set = Place(N) - Place(SliceSet);
    for ∀ p ∈ Uncovered_Place_Set do {
        slice = find_minimally_connected(SliceSet, p);
        slice = slice ∪ { p };
    }
}
```

Figure 3: The Slicing Algorithm for Petri net models

Figure 3 shows a slicing algorithm which slices a Petri net model into a set of Petri net slices using minimal invariants. At first, S-invariants are computed and then the minimal invariants are selected among S-invariants (*find_minimal_invariants*). In the minimal invariant set, the algorithm chooses an invariant which has the minimal number of elements (*find_smallest_invariant*) and adds it into *SliceSet* until *SliceSet* covers all the places in *N* (*Place*(*N*) == *Place*(*SliceSet*)) or there exists no minimal invariant which includes a new place (*Place*(*SetofInvariant*) $\subseteq$ *Place*(*SliceSet*)). If the minimal invariant set becomes empty without covering all the places in *N*, for each uncovered place, it should be added into a slice to which it is connected by a minimal number of transitions (*find_minimally_connected*). In this manner, the slicing algorithm ensures that every place in *N* belongs to some slices.

The complexity of the slicing algorithm is mainly dependent on three time-consuming procedures: *find_minimal_invariant*, *find_smallest_invariant*, and *find_minimally_connected*. The time complexity of *find_minimal_invariant* is the same as that of finding S-invariants, which is described in the previous section. Finding a smallest invariant is similar to sorting invariants in criteria of the number of elements. In *find_minimally_connected* procedure, one transition-connectable places, which are connected to the uncovered place through a transition, are checked if they are members of any SliceSet. If one of them is included in some SliceSet, the uncovered place is added into the SliceSet. Otherwise, the procedure proceeds to two transitions-connectable places. Therefore, the time complexity of the slicing algorithm is $O(n^3)$ for n number of places in a P/T net.

A modular Petri net, *Petri net slices*, is defined using *SliceSet* as follows.

Definition 3.1 *Petri net slices*
Let $N = (P, T, F, W, M)$ *be an original P/T net where P represents a set of places, T is a set of transitions, F is a set of arcs, W is weight functions of arcs, and M is the initial marking. And let* $SliceSet = \{P\_Slice_i \mid i = 1 \ldots n\}$ *be a set of place sets which are obtained by the slicing algorithm. Petri net slices are defined as* $\{Petri\ net\ slice_i \mid i = 1 \ldots n\}$, *where* $Petri\ net\ slice_i = (P_i, T_i, F_i, L_i, W_i, M_i)$ *satisfies the following:*

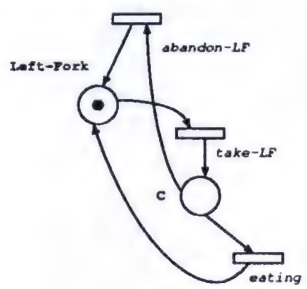
- $P_i = P\_slice_i$,
- $T_i = \{ t \in T \mid \forall p \in P_i, \exists(p, t) \in F \text{ or } \exists(t, p) \in F \}$,

- $F_i = \{ (p, t) \in F \text{ or } (t, p) \in F \mid \forall p \in P_i, \exists t \in T_i \}$,
- $L_i : T_i \rightarrow \Sigma^+$ *is a label function of transitions, where* Σ *is a character set. We assign the name of a transition (e.g. tl) in N to a label of t, that is,* $L_i(t) = tl$,
- $W_i(p) = W(p)$, *and* $M_i(p) = M(p)$, $\forall p \in P_i$.

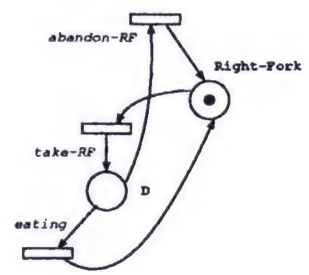
In a single Petri net model, a transition t is enabled if the places connected by incoming arcs of t (denoted as $^\bullet t$) have sufficient tokens. In Petri net slices, shared transitions whose labels appear in several slices are fired by synchronising the same labels. Following defines enabledness of a transition in Petri net slices.

- $L_i(t)$ is not shared: a transition t is enabled if the precondition of t is satisfied.
- $L_i(t)$ is shared among several Petri net slices: a transition t is enabled if t is enabled in all the slices which have the label of t.

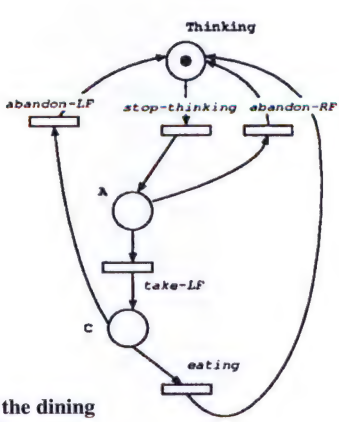
Figure 4 shows the result of applying the slicing algorithm to the model in Figure 2. The slices shown in Figure 4(a) and (b) are S-invariants containing Left-Fork place and Right-Fork place, respectively. Figure 4(c) represents the behaviour for taking the left fork or abandoning the taken right fork. Figure 4(d) represents the behaviour for taking the right fork. In the dining philosopher model, there exists no uncovered place.



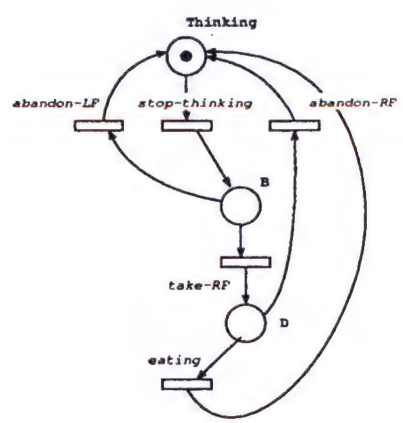
(a) slice 1



(b) slice 2



(c) slice 3



(d) slice 4

Figure 4:
Petri net slices of the dining philosopher problem

In order to show that the original model and the Petri net slices obtained by applying the slicing algorithm have equivalent behaviours, we define the concept of behavioural equivalence of Petri net models. In the definition, we concern only observational behaviour, where the system behaviours are viewed as black boxes by considering only externally visible behaviours without considering the internal structures of the models (Milner, 1989).

Definition 3.2 *Behavioural Equivalence of two P/T nets (P ~ Q) :*
P and Q are behaviourally equivalent iff there exists an one-to-one mapping between the reachability graphs of P and Q.

Theorem 3.1 *Let N be a Petri net and S be the Petri net slices obtained by the slicing algorithm. The original model N and the slice model S are behaviourally equivalent (N ~ S)*

Mapping of RGs can be characterised by mapping between occurrence sequences of corresponding transitions. And transition occurrences are determined by pre/post-places of transitions. Therefore, equivalent RGs have the same initial marking and the same pre/post-conditions of the corresponding transitions.

Since the slicing algorithm preserves input and output flows information of places, all the corresponding places of N and S have the same initial tokens and the same behaviours. Hence, we show only the equivalence of pre/post-conditions of t_N and its corresponding label tl_S.

(Proof)
Let $N = (P,T,F,W,M)$ be a Petri net model and $S = \{C_{n_i} | i = 1...n\}$, where $C_{n_i} = (P_i, T_i, F_i, L_i, W_i, M_i)$, which is obtained by applying the slicing algorithm to N, be Petri net slices.

Let tl_S be a uniquely corresponding label of a transition t_N. Since a transition label can have multiple instance transitions in several C_{n_i}, pre/post-place sets of label tl_S are differently denoted as ${}^\circ tl_S$ and $tl_S{}^\circ$, and defined by the unions of pre/post-place sets (${}^\bullet t_{C_{n_i}}$ /$t^\bullet_{C_{n_2}}$) of multiple instance transitions $t_{C_{n_i}}$.

We will first show that ${}^\bullet t_N$ is equal to ${}^\circ tl_S$ by showing ${}^\circ tl_S \subseteq {}^\bullet t_N$ and ${}^\bullet t_N \subseteq {}^\circ tl_S$ on the assumption that the label tl_S is the name of t_N.

Case 1: ${}^\circ tl_S \subseteq {}^\bullet t_N$

For each C_{n_i}, if there exists a transition $t_i \in T_i$ such that $L_i(t_i) = tl_S$, it must be an instance of t_N. Since incoming arc information of t_i was derived from the arc information of t_N through the slicing algorithm, the pre-places of t_i should be a subset of the pre-places of t_N. Therefore, ${}^\circ tl_S$ is a subset of ${}^\bullet t_N$ since ${}^\bullet t_i$ is a subset of ${}^\bullet t_N$ for all C_{n_i}.

Case 2: ${}^\bullet t_N \subseteq {}^\circ lt_S$

In order to show the subset relation of two sets, we proof the following equivalent implication relation, $\exists p \in {}^\bullet t_N \Rightarrow p \in {}^\circ tl_S$. For each $p \in {}^\bullet t_N$, the place p should be included in one or more C_{n_i}. Assume that the place p is contained in C_{n_2}. Then a transition t_2 such that $L_2(t_2) = tl_S$ should be included in T_2 since the incoming arc information (t_N) of p should be included in C_{n_i}. That is, $p \in {}^\bullet t_2$. Therefore, p is also included in ${}^\circ tl$ since ${}^\bullet t_2$ is a subset of ${}^\circ tl$.

Similarly, the equation $t^\bullet_N = tl^\circ_S$ can be proved.

4. COMPOSITIONAL REACHABILITY ANALYSIS OF PETRI NET SLICES

Compositional analysis can be applicable to a system model which is composed of several modules. After generating the reachability graph of each module and analysing the local properties of the modules, the local reachability graphs are reduced and composed to analyse the global properties of the system. Compositional generation and analysis of reachability graphs are well described in Yeh's (1993) work.

Figure 5 describes the composition procedure of two reachability graphs (RGs). Figures 5(a) and (b) represent the reachability graphs of slice1 and slice3 shown in Figure 4, respectively. In combining two RGs, the labels of transitions have an important role. Transitions with the same label can be synchronously performed only when they are ready in each RG. On the other hand, when non-shard transitions are ready in its own RG, they can occur regardless of the states of other RGs. In Figure 5(c), for example, a shared transition, *take-LF*, only occurs at state RS1.4, which indicates that slice1 is at state RS1 and slice3 is at state RS4.

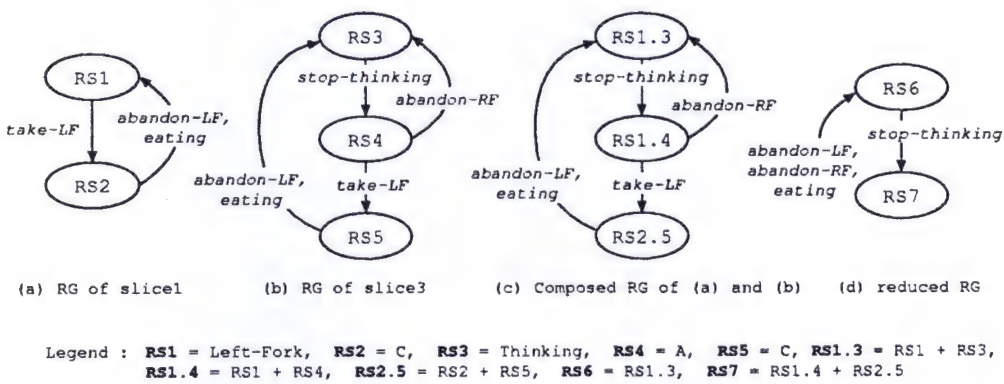


Figure 5: Composition procedure of two reachability graphs

Among the transition labels in Figure 5(c), the label *'take-LF'* does not appear in other RGs of slice2 and slice4 as shown in Figure 4. These transitions are called 'local' transitions. Since local transitions have no impacts on interactions with other slices, they can be reducible. By reducing the *take-LF* transition, RS1.4 and RS2.5 states are merged to RS7 state, as shown in Figure 5(d).

Reachability analysis of Petri net slices is performed on the hierarchical reachability graph of slices. Liveness property of transitions should be hierarchically checked since the behaviours of shared transitions are characterised by those of other slices. On the other hand, Boundedness property of places are checked in each reachability graph of slices. The characteristic of S-invariants is that all the outgoing and incoming arc information of a place are preserved in each S-invariant. Therefore, the global behaviours of a place can be predictable in local slices. If there is an ω-place(a place which can have an unlimited number of tokens) in the reachability graph of a slice, we can find that the place is globally unbounded.

5. CASE STUDY

In this section, we apply Petri nets slices to dining philosophers problems and feature interaction problems of telecommunication systems (Keck and Kuehn, 1998). As an ideal concurrent example, we choose dining philosophers problems to illustrate the efficiency of Petri net slices in handling

concurrency during generating reachability graphs. And feature interaction problem is chosen to show applicability of our approach to a practical example and to briefly mention how to use Petri net slices in analysing system properties such as unboundedness and nondeterminism.

5.1 Dining Philosophers problems

P/T nets, Modular Petri nets (Notomi and Murata, 1994), and Petri net slices approaches are used to describe dining philosophers problems. The P/T net model shown in Figure 1 is extended for describing several philosophers. In Modular Petri net approach, although a modeller can divide an entire problem into modules by his/her own partitioning criteria, we assume that the model is composed of subnets of each philosopher and each fork. Petri net slices are obtained by applying the slicing algorithm to the P/T net model. A single philosopher case has four slices. In generating compositional reachability graphs, the number of system states varies on the composition orders (Yeh, 1993). In order to fairly compare Modular Petri nets and Petri net slices approaches, we apply the same composition order when generating compositional reachability graphs. Table 1 shows the numbers of reachable states and state transitions of three approaches.

phil. #	P/T nets	Modular PN		PN slices	
		# modules	# states(trans.)	# modules	# states(trans.)
1	5(8)	3	15(24)	4	18(29)
2	18(46)	4	20(40)	8	38(61)
3	76(291)	6	55(134)	12	58(93)
4	322(1644)	8	77(195)	16	78(157)
5	1364(8705)	10	99(256)	20	98(157)
6	5778(44250)	12	121(317)	24	118(189)
7	24476(218687)	14	143(378)	28	138(221)
8	103682(1058712)	16	165(439)	32	158(253)
9	439204(5045373)	18	187(500)	36	178(285)
10	-	20	209(561)	40	198(317)
20	-	40	429(1171)	80	398(637)
30	-	60	649(1781)	120	598(957)
50	-	100	1089(3001)	200	998(1597)

Table 1: The numbers of reachable states and state transitions for dining philosopher models

As shown in Table 1, while the numbers of reachable states and state transitions grow exponentially in P/T nets, the numbers in Modular Petri nets and Petri net slices grow slowly. That is, the compositional approach is an effective method to reduce state explosion. The number of modules in Petri net slice approach is larger than that of Modular Petri nets. That is, Petri net slice approach can support finer granularity of concurrency. Therefore, as the number of philosophers grows, the number of reachable states in Petri net slices is a little less than that in Modular Petri nets, and large differences are shown in the numbers of state transitions.

5.2 Feature Interaction Problem

In telecommunication systems, the term *feature interaction* refers to situations, where different service features or instances of the same service feature affect each other. This can take place within the same service as well as between features of different services. As the increasing demand for new telecommunication services has led to a rapidly growing number of new services, as well as to the

enhancement of existing services with new service features, there are strong demands on formal approaches that describe the behaviours of service features and detect undesirable interactions among them.

In this section, we describe basic call processing (BCP) and call forwarding (CF) service features by P/T nets and analyse the interaction between BCP and CF service features by using Petri net slices.

There are six representative use cases dealing with basic call processing in the telecommunication software, where the first three use cases are extracted in a caller's viewpoint and the others are described in a callee's viewpoint. CF service feature has two use cases; one is to subscribe or unsubscribe the CF service feature and the other represents the functionality of call forwarding. The following shows eight use cases corresponding to BCP and CF service feature.

- use case 1 : a caller calls, talks with, and hangs up.
- use case 2 : a caller dials a number, but no one answers.
- use case 3 : a caller calls, but the destination line is busy.
- use case 4 : a callee picks up phone, talks with and hangs up.
- use case 5 : before picking up the phone, ringing stops.
- use case 6 : while talking with someone, another call arrives.
- use case 7 : an incoming call is forwarded to the CF-registered destination.
- use case 8 : a user can subscribe or unsubscribe CF service feature.

Figure 6 shows a P/T net model representing eight use cases. The shaded part represents use case 7, which represents the call forwarding service feature.

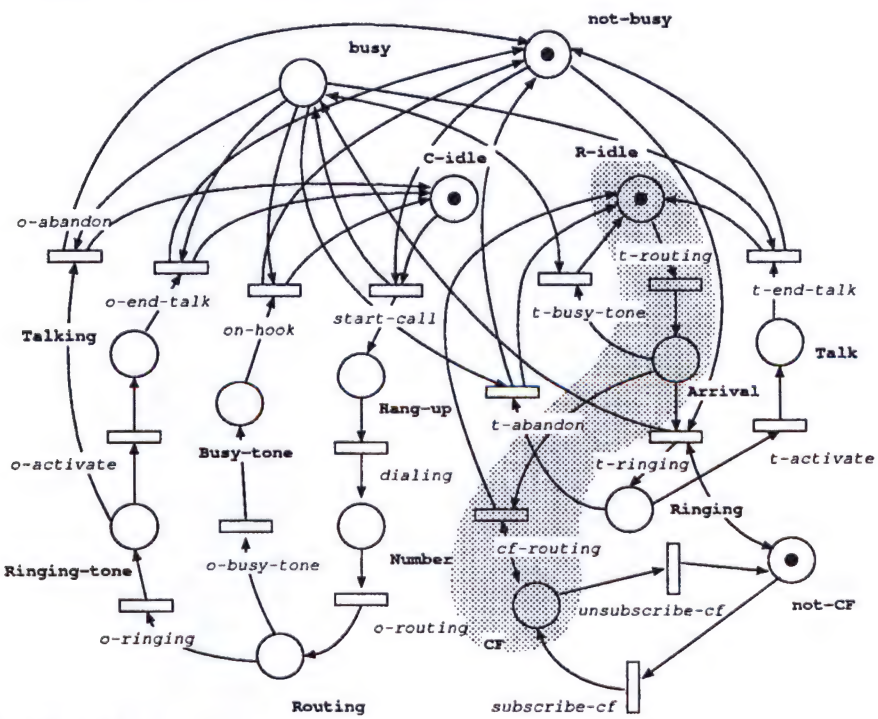


Figure 6:
A P/T net for BCP and CF

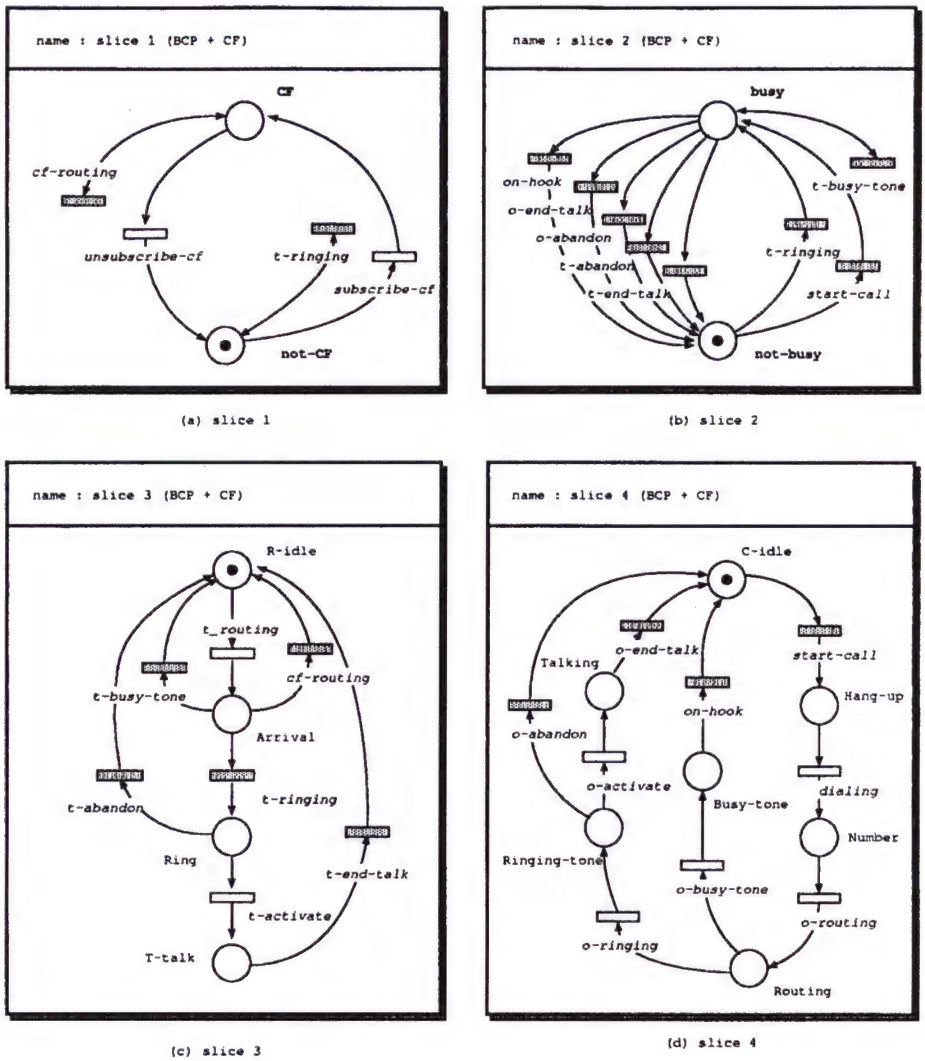


Figure 7: Slice sets of the BCP and CF model

After applying the slicing algorithm to the P/T net model for BCP and CF, we obtain the Petri net slices shown in Figure 7. The slice of Figure 7(a) is related with forwarding functionality, and Figure 7(b) shows a slice for *busy* and *not-busy* toggling conditions. Figures 7(c) and (d) represent the behaviours of a callee and a caller, respectively.

In Figure 7, each slice has no unbounded place. Since boundedness property of places can be locally checkable as mentioned in the previous section, there is no unbounded place in the BCP and CF model. Nondeterminism among transitions can be checked on compositional reachability tree. In the reachability graphs of Petri nets slices, there may be many locally-nondeterministic situations since the interactions with other slices are not considered. In the composed reachability graphs, however, these nondeterministic situations may be resolved by synchronising shared transitions.

In Figure 7(c), there are two nondeterministic situations: {*t-abandon, t-activate*} and {*t-busy-tone, t-ringing, cf-routing*}. A nondeterministic situation between *t-abandon* and *t-activate* is an external caller's choice inherited from BCP model, which is not related with interactions among BCP and CF. Among nondeterministic situations in {*t-busy-tone, t-ringing, cf-routing*}, the cases {*t-ringing, cf-routing*} and {*t-ringing, t-busy-tone*} are resolved by merging with RGs of slice1 and slice2, respectively. However, the nondeterministic situation of *t-busy-tone* and *cf-routing* is not resolved in composed RGs, which means that BCP's busy-tone signaling and CF's cf-routing can nondeterministically occur when another call arrives during conversation. This nondeterminism is caused by the modeler's mistake, that is, he forgot adding *not-CF* condition to *t-busy-tone* transition in Use Case 6 during introducing the call forwarding SF into BCP model.

As shown in the above example, the role of Petri nets slices approach is to divide a huge P/T net model into several manageable modules for applying compositional reachability analysis techniques. In Petri net slices, Petri net model analysers can focus on each local slice and incrementally proceed the analysing procedure to composed models for verifying interactions of slices.

6. CONCLUSIONS AND FUTURE WORK

We proposed the Petri nets slice approach in order to partition huge P/T net models into manageable modules, so that the partitioned model can be analysed by compositional reachability analysis technique. The Petri net slice approach can be used for avoiding state explosion in reachability analysis of a huge P/T net model and for efficiently analysing system properties in compositional approach. Our approach is well suited to concurrent or distributed systems, which are described by P/T nets or P/T nets-transformable high-level Petri nets.

As future work, we have a plan to extend Petri net slice concept to be directly applicable to Colored Petri nets, which have been frequently used in practical applications. We have implemented the slicing algorithm in the form of a text-based package. We have a plan to connect this package with a new graphical Petri net tool or existing Petri net tools such as Design/CPN (University of Aarhus, 1996). And we will enhance the compositional reachability analysis techniques on the Petri net slices and implement supporting packages.

REFERENCES

SUZUKI, T., SHATZ, S.M., and MURATA T. (1990): A Protocol Modeling and Verification Approach Based on a Specification Language and Petri Nets, *IEEE Trans. on Software Engineering*, 16(5), May: 523-536.

GHEZZI, C., MANDRIOLI, D., MODASCA, S., and PEZZE, M. (1991): A Unified High-Level Petri Net Formalism for Timed-Critical Systems, *IEEE Trans. on Softw. Eng., 17(2)*: 160-172.

BUCCI, G. and VICARIO, E. (1995): Compositional Validation of Time-Critical Systems Using Communicating Time Petri Nets, *IEEE Trans. on Softw. Eng., 21(12)*: 969-992.

JENSEN, K. (1992): *Coloured Petri Nets: Basic Concepts, Analysis methods and Practical Use. Volume 1*, Springer-Verlag.

BATTISTON, E. and CINDIO, F.D. (1993): Class Orientation and Inheritance in Modular Algebraic Nets, *Proc. of IEEE Conf. on System, Man, and Cybernetics, 2*: 712-723.

PERKUSICH A. and FIGUEUREDI J. (1997): G-Nets: a Petri Net Based Approach for Logical and Timing Analysis of Complex Software Systems, *J. of Syst. and Softw., 39:* 39-59.

LEE, W.J., CHA, S.D., and KWON, Y.R. (1998): Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering, *IEEE Trans. on Softw. Eng., 24(12):*1115-1130.

MURATA, T. (1989): Petri nets : Properties, Analysis and Applications, *Proc. of the IEEE, 77(4):*541-580.

ESPARZA, J. (1993): Model Checking using net unfoldings, *TAPSOFT '93(LNCS #668),* 613-628.

JORGENSEN, J.B. and KRISTENSEN, L.M. (1997): Verification of Coloured Petri Nets Using State Spaces with Equivalence Classes, *Proc. of the Workshop on Petri Nets in Sys. Eng.(PNSE'97),* September.

DAMM, W., DOHMEN, G., GERSTNER, V., and JOSKO, B. (1989): Modular Verification of Petri Nets: The Temporal Logic Approach, *LNCS #430,* 180-207.

VALMARI, A. (1990): Compositional State Space Generation, *Proc. of Appl. and Theory of Petri Nets '89.*
NOTOMI, M. and MURATA, T. (1994):Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent-Software Analysis, *IEEE Trans. on Softw. Eng., 22(5)*: 325-336.
CHRISTENSEN, S. and PETRUCCI, L. (1995): Modular State Space Analysis of Coloured Petri Nets, *Proc. of Appl. and Theory of Petri Nets '95(LNCS #935),* 201-217.
VALMARI A. (1994): Compositional Analysis with Place-Bordered Subnets, *Proc. of Appl. and Theory of Petri Nets '94(LNCS #815),* 531-547.
SCHREIBER G. (1995): Functional Equivalences of Petri Nets, *Proc. of Appl. and Theory of Petri Nets '95(LNCS #935),* 432-450.
REISIG, W. (1985): *Petri Nets:An Introduction,* Springer-Verlag.
YEH, W.J. (1993): *Controlling State Explosion in Reachability Analysis,* PhD. Thesis, Purdue University, December.
MILNER, R. (1989): *Communication and Concurrency,* Prentice-Hall.
KECK D.O. and KUEHN P.J. (1998): The Feature and Service Interaction Problem in Telecommunication Systems: A Survey, *IEEE Trans. on Softw. Eng., 24(10):* 779-796.
UNIVERSITY OF AARHUS (1996): *Design/CPN User's Guide : Version 3.0,* University of Aarhus.

BIOGRAPHICAL NOTES

Woo Jin Lee received his B.S. degree in computer science from KyungPook National University, Korea, in 1992 and M.S. and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Korea, in 1994 and 1999. He is currently a senior member of engineering staff in Electronics and Telcommunication Research Institute (ETRI) Computer & Software Technology Laboratory (CSTL).

His research interests include requirements engineering, safety-critical systems, component-based software engineering and process modelling.

Sung Deok (Stephen) Cha received B.S., M.S. and Ph.D. degrees in information and computer science from the University of California, Irvine in 1983, 1986 and 1991. He taught at California State University of Borthridge and Long Beach from 1985 to 1990. From 1990 to 1994, he was a member of the technical staff at the Hughes Aircraft Company, Ground Systems Group and the Aerospace Corporation where he worked on various projects on software safety and computer security. In 1994, he became a faculty member of the KAIST Computer Science department.

His research interests include software safety, formal methods and computer security.

Yong Rae Kwon received B.S. and M.S. degrees in physics from Seoul National University, Korea, in 1969 and 1971 respectively, and a Ph.D. in physics from the University of Pittsburgh in 1978. He taught as an instructor at Korea Military Academy from 1971 to 1974. He was on the technical staff of Computer Science Corporation from 1978 through 1983 working on the ground support software systems for NASA's satellite projects. He joined the Faculty of Computer Science of KAIST in 1983.

His research interests include verification of real-time parallel software, object-oriented technology for real-time systems and quality assurance for highly dependable software.

Heung-Nam Kim received a B.S. degree in electronic engineering from Seoul National University, Korea, in 1980, an M.S. in computer science from Bell State University in 1989, and Ph.D. in computer science from Pennsylvania State University in 1996.

He has been working for ETRI CSTL since 1983 and is currently the leader of the embedded software research team.

His research interests include video compression algorithm, real-time systems, software engineering and distributed multimedia systems.

DALHOUSIE UNIVERSITY

Faculty Positions in Computer Science

Dalhousie University invites applications for tenure track positions at all levels within the new Faculty of Computer Science. The Faculty has a combined complement of 23 faculty positions and approximately 600 undergraduate majors and 135 master's and doctoral students. The expansion and development of the Faculty is a priority for the University.

The Faculty will continue to experience considerable growth over the next few years in all aspects: faculty complement, student enrolment, funding levels and facilities. The Faculty recently moved to a new building and has secured significant infrastructure funding for 2000–2001. New research laboratories are planned and initiatives involving multidisciplinary research projects with university and industrial partners are under development. As an example a new Master of Electronic Commerce degree is now offered in collaboration with the Faculties of Law and Business.

Applicants should have a Ph.D in Computer Science or related area and show a strong commitment to and aptitude for teaching and research. Rank and salary will be commensurate with qualifications. The major research foci of the Faculty are Network Centric Computing and Software Engineering. Individuals with expertise in these and related areas, such as networking, HCI, distributed applications, etc., are especially encouraged to apply. Successful candidates will be required to teach in both the undergraduate and graduate programs, to establish research programs, to contribute to the administration of the Faculty and will also be encouraged to establish significant industrial connections.

Dalhousie University is located in Halifax, Nova Scotia, which is the largest city in Atlantic Canada and affords its residents outstanding quality of life.

Applications should include a curriculum vitae and the names and complete addresses of three references. They should be addressed to:

The Chair, Appointments Committee
Faculty of Computer Science
Dalhousie University
6050 University Avenue
Halifax, Nova Scotia, Canada B3H 1 W5

E-mail: appointments@cs.dal.ca

URL: www.cs.dal.ca

Applications will be reviewed on an ongoing basis until all available positions are filled.

In accordance with Canadian immigration requirements, priority will be given to Canadian citizens and permanent residents of Canada. Dalhousie University is an Employment Equity and Affirmative Action employer. The University encourages applications from qualified Aboriginal peoples, persons with a disability, racially visible peoples and women.

THE FLINDERS UNIVERSITY OF SOUTH AUSTRALIA

FACULTY OF SCIENCE AND ENGINEERING
SCHOOL OF INFORMATICS AND ENGINEERING

SENIOR LECTURER/LECTURER

INFORMATION TECHNOLOGY,
SOFTWARE ENGINEERING, COMPUTER SCIENCE
(Continuing – Three positions available)

Lecturer: $48, 494 – $57, 586 pa • Senior Lecturer: $59, 405 – $68, 497 pa • Ref 00165

The recently formed School of Informatics and Engineering has identified Software Engineering and Information Technology as key strategic directions and is looking to strengthen and expand its activities in these areas. The School also recognises the need to support core Computer Science that underpins these strategic directions. To support these strategies, applications are invited for three Lecturer or Senior Lecturer positions within these areas. Applicants will need to establish how their teaching and research interests integrate with the strategic directions.

The School offers teaching and research programs at all levels in Computer Science, Information Technology and Software Engineering as well as contributing to many other awards. The School's highly successful Bachelor of Information Technology has a particular focus on the design, implementation and management of enterprise-wide information systems particularly Web-based information systems. In Software Engineering, there is a particular focus on object-oriented software design and development to complement the use of Java as the main programming language. The School recognises the need to maintain a strong core Computer Science curriculum and is particularly interested in suitably qualified individuals who can contribute to this curriculum in the areas of computer organisation, operating systems and network systems.

Research in Information Technology, Software Engineering and Computer Science at Flinders can be broadly defined as applied research related to internet-based and data-rich information technology systems. Much of this research is highly cross-disciplinary, collaborative and emphasizes industry-focussed applications areas.

Applicants for all of these positions should hold a PhD degree in a relevant area, or have equivalent postgraduate training and experience. A successful applicant for appointment at the Senior Lecturer level will have an established teaching record and substantial research record or other relevant experience. The positions are available from January 2001 and are continuing after the normal probationary period. There is provision for a retention allowance. Packages to support on-going professional development may also be negotiated.

Enquiries should be directed to: Janet Verbyla, Deputy Head, School of Informatics and Engineering, phone +61-8-8201-2959, email to janet@infoeng.flinders.edu.au, or by fax to +61-8-8201-3626. Applicants can obtain the information kit via the web-site, www.infoeng.flinders.edu.au, or from Sharon Appleyard, Manager, School of Informatics and Engineering, phone +61-8-8201-2072, email to sharon@infoeng.flinders.edu.au, or by fax to +61-8-8201-2904.

Applicants must address the selection criteria detailed in the information kit, quote the reference number, clearly indicate the position(s) for which they wish to be considered and give the names, addresses and facsimile numbers/email addresses of three referees of whom confidential enquiries may be made. Written applications should be lodged, in duplicate, with the Manager, Human Resources, The Flinders University of South Australia, GPO Box 2100, Adelaide SA 5001, or fax 8201 3131. Applications will be accepted until the positions are filled. **To be considered in the initial selection round, applications need to be received by Tuesday 3 October 2000.**

The University reserves the right not to make an appointment or to invite applications.

Applications from suitably qualified women are particularly welcome

EQUAL OPPORTUNITY IS UNIVERSITY POLICY

NOTES

NOTES

Submission Guidelines

The Journal encourages submission of innovative and original articles in all areas of Information Technology including Computer Science, Software Engineering, Information Systems, Computer Systems and Information Engineering and Telecommunications.

The following forms of article are published:

- Full articles examining original research and/or application;
- Short articles describing original research and/or application or commenting on relevant legal, political or technical innovations;
- Survey or tutorial articles describing new directions in Information Technology or which are relevant to the practical application of fundamental research;
- Book Reviews;
- Announcements; and
- News Briefs.

Full details can be found at the Journal home page: www.jrpit.flinders.edu.au

Subscription Guidelines

The annual subscription for the Journal is

Individuals	
Australian Computer Society members	Free
Delivery address in Australia, incl. postage and GST when applicable	$60
Overseas, incl. postage.	$75
Institutions	
Australia and Overseas, including postage and GST where and when applicable.	$90

All subscriptions to the Journal are payable in advance and should be sent (in Australian Currency) to the address below.

Advertising Information

The Journal accepts advertisements in the following categories:

- IT-related career opportunities
- IT-oriented training and education courses;
- IT-related conferences and workshops.

Rates are as follows:

Full Page	$1,500
Half Page	$750
Quarter Page	$375
By line	$100 for heading plus up to six lines of text (40 characters per line).
	$50 for each additional three lines, or part thereof.

Full details can be found at the Journal home page http://www.jrpit.unisa.edu.au.

Bibliographic Information

Journal of Research and Practice in Information Technology – *J.Res.Prac.Inf.Tech.*
formerly the Australian Computer Journal (ISSN 004-8917)

ISSN	1443-458X
Circulation	13,374 (December 1999)
Publisher	Australian Computer Society Inc. PO Box Q534, QVB Post Office Sydney 1230, New South Wales AUSTRALIA
Production Management	Associated Business Publications Pty. Ltd. P.O. Box 3267, Umina, NSW 2257, Australia
Printer	Mailing and Print Services Pty Ltd 8 Aquatic Place, Frenchs Forest, NSW 2086, Australia
Cover Design	Modern Planet Design (08) 8340 1361 • modern@webmedia.com.au

Contents

75 Partitioning and Allocation of Objects in Distributed Application Development
G.C. Low and G. Rasmussen

107 Applying the ACS Code of Ethics
O.K. Burmeister

121 Partitioning Number Sequences into Optimal Subsequences
J. Zobel and P. Dart

131 A Slicing-Based Approach to Enhance PetriNet Reachability Analysis
W.J. Lee, H.N. Kim, S.D. Cha and Y.R. Kwon

Special Features

73 Editorial

A publication of the Australian Computer Societ
ISSN 1443- 458X